

版权注意事项：

- 1、书籍版权归作者和出版社所有
- 2、本PDF仅限用于个人获取知识，进行私底下的知识交流
- 3、PDF获得者不得在互联网上以任何目的进行传播
- 4、如觉得书籍内容很赞，请购买正版实体书，支持作者
- 5、请于下载PDF后24小时内删除本PDF。

高性能Linux 服务器构建实战

系统安全、故障排查、自动化运维与集群架构

高俊峰 著

Build High Performance Linux Servers
System Security, Troubleshooting, Automated
Operations, Cluster Architecture

- 畅销书《高性能Linux服务器构建实战：运维监控、性能调优与集群应用》姊妹篇，资深运维专家与架构师多年经验结晶
- 以实际生产环境为背景，从系统安全、故障排查、自动化运维、集群架构4个维度讲解构建大规模和高性能Linux服务器集群所需技术、工具、方法及技巧



机械工业出版社
China Machine Press

作者简介

高俊峰（南非蚂蚁）资深运维专家、系统架构师、DBA和技术顾问，从事Linux/Unix服务器系统的架构、运维和管理多年。擅长大规模服务器集群的运维和管理，在故障诊断与排除、自动化运维、安全运维、性能调优、虚拟化等方面积累了大量的实战经验。国内知名IT技术社区51CTO的博客专家和ChinaUnix论坛高可用集群板块的版主，同时还活跃于ITPUB等技术社区，在社区和论坛里发表了大量技术文章，深受欢迎。此外，他还著有畅销书《循序渐进Linux》和《高性能Linux服务器构建实战：运维监控、性能调优与集群应用》，后者是本书的姊妹篇，是Linux运维领域公认的经典著作。

作者博客：

<http://ixdba.blog.51cto.com>

<http://www.ixdba.net/>

关注微信公众号ednote进入“第三极社区”，与作者和其他读者互动。

高性能Linux 服务器构建实战

系统安全、故障排查、自动化运维与集群架构

Build High Performance Linux Servers
System Security, Troubleshooting, Automated
Operations, Cluster Architecture

高俊峰 著



机械工业出版社
China Machine Press

图书在版编目 (CIP) 数据

高性能 Linux 服务器构建实战：系统安全、故障排查、自动化运维与集群架构 / 高俊峰著.
—北京：机械工业出版社，2014.7（2016.12 重印）
（Linux/Unix 技术丛书）

ISBN 978-7-111-47249-0

I. 高… II. 高… III. Linux 操作系统 IV. TP316.89

中国版本图书馆 CIP 数据核字（2014）第 148500 号

本书是 Linux 运维领域畅销的、公认的经典著作《高性能 Linux 服务器构建实战：运维监控、性能调优与集群应用》的姊妹篇，它从安全运维、运维故障排查、自动化运维、集群架构四个维度讲解了构建大规模和高性能 Linux 服务器集群所需要的技术、工具、方法和技巧，二者一脉相承，互为补充，内容涵盖了运维工程师构建高性能服务器需要掌握的各种知识。本书在内容上继承了其姊妹篇中被读者认可的诸多优点：实用（以实际生产环境为背景）、实战（包含大量案例）、易懂，同时也改进了读者反馈的不足之处。

全书共 14 章，分为四个部分：安全运维篇（第 1~3 章）主要讲解服务器安全运维、网络安全运维和数据安全运维的关键技术和方法；运维故障排查篇（第 4~5 章）从服务器系统和应用软件两个维度讲解运维故障的排查思路，以及常见的和经典的运维故障的解决方案；自动化运维篇（第 6~9 章）主要讲解海量主机的自动化部署和配置、自动化监控、分布式监控等大规模集群运维所需的各种工具（pssh、pdsh、mssh、Ganglia、nagios、Centreon 等）和方法；集群架构篇（第 10~14 章）讲解如何构建和优化基于 Web 和 MySQL 数据库的高性能集群和高可用的负载均衡集群。

高性能 Linux 服务器构建实战： 系统安全、故障排查、自动化运维与集群架构

高俊峰 著

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码：100037）

责任编辑：杨福川

责任校对：董纪丽

印 刷：北京市荣盛彩色印刷有限公司

版 次：2016 年 12 月第 1 版第 7 次印刷

开 本：186mm×240mm 1/16

印 张：26.25

书 号：ISBN 978-7-111-47249-0

定 价：79.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

客服热线：(010) 88379426 88361066

投稿热线：(010) 88379604

购书热线：(010) 68326294 88379649 68995259

读者信箱：hzit@hzbook.com

版权所有·侵权必究

封底无防伪标均为盗版

本书法律顾问：北京大成律师事务所 韩光 / 邹晓东

为什么要写这本书

随着虚拟化、云计算时代的来临，Linux 得到迅猛发展，在服务器领域已经占据半壁江山，而基于 Linux 的运维也面临新的挑战：面对越来越复杂的业务，越来越多样化的用户需求，不断扩展的应用需要越来越合理的模式来保障 Linux 灵活便捷、安全稳定地持续提供服务，这种模式中的保障因素就是 Linux 运维。从初期的几台服务器发展到庞大的云计算数据中心，单靠人工已经无法满足在技术、业务、管理等方面的要求，那么标准化、自动化、架构优化、过程优化等降低运维成本的因素越来越受到人们所重视。其中，以自动化运维代替人工操作为出发点的诉求得到广泛研究和应用。

2012 年我完成了基于 Linux 运维的开源软件应用作品《高性能 Linux 服务器构建实战：运维监控、性能调优与集群应用》^①。此书出版后，得到了很多同行的认可，这也成为鼓励我继续写作的动力，但是仅仅通过一本书没办法将运维工作中的所有内容完全展开讲述，因此，本书应运而生。

目前市场上关于 Linux 运维管理的书籍有很多，但是普遍存在的问题是模式单一，要么只讲基础理论和系统命令，要么侧重粘贴代码，要么介绍软件的安装与配置，这种模式带有很大的实验性质，并没有介绍生产环境中的实战应用和经验技巧。

本书针对这些问题，从基础入手，再进行深入研究，同时结合实际的应用案例进行由点到线及面、由浅入深的讲述。本书秉承了实战、实用、通俗、易懂的特点，在内容上十分注重实战化，从运维的多个方面以真实的生产环境介绍运维工作中的方方面面，理论介绍结合实际应用贯穿全书，通过学习真实案例，读者可以深入、迅速地掌握 Linux 运维技术的各种经验和技巧，从而真正提高实践能力。

① 本书由机械工业出版社出版，ISBN:978-7-111-36695-9。

图书在版编目(CIP)数据

本书分为四篇，以 Linux 运维平台下的开源应用软件为中心，涉及 Linux 运维的各个方面，主要从安全运维、运维故障排查、自动化运维、集群架构四个方面展开介绍。其中，安全运维篇是本书的一大亮点，它是运维中很容易忽略但又非常重要的一个组成部分，放眼同类图书，介绍安全运维方面内容的并不多。运维故障排查篇是对生产环境实际案例与经验技巧的总结，通过讲述实际案例，使读者有身临其境的感觉，并从中获取处理问题的思路和技巧。随着云计算、虚拟化在企业的普及和深入，大规模的服务器集群运维是必然趋势，这就促使 Linux 运维向自动化方面发展，即自动化运维，本书自动化运维篇主要介绍了海量主机的自动化部署、配置工具，接着介绍了运维的核心：自动化监控、分布式监控的应用，这些内容在大规模集群运维下是非常有用的，这也符合运维日益发展的需要。而最后的集群架构篇主要介绍当今流行的集群架构方案和集群应用软件，应该说是本书内容的综合和深入，也是本书介绍的重点。

本书是笔者多年实践工作的经验和总结，全书贯穿了由点及线、由线及面的学习方法，既可以让初学者参考学习，也可以帮助有一定基础的中高级 Linux 运维管理人员进阶学习，使不同层次的读者都能从本书受益。

读者对象

本书适合的阅读对象有：

- 中高级 Linux 运维管理人员
- Linux 系统工程师
- 系统集成商
- 解决方案构架师
- 所有开源爱好者

如何阅读本书

本书最大的特点是注重实践、理论与实际相结合，在讲述完一个知识点后，一般都附有实例作为对知识的补充，并且每个章节都是一个独立的知识块，读者可以选择从中间阅读，也可以从第 1 章依次阅读。纵观全书，每个知识的介绍都由浅入深，由点及面。

本书主要分为四篇，总计 14 个章，基本结构如下。

安全运维篇（第 1 章至第 3 章）

安全运维篇介绍了系统运维中安全应用的三个方面：服务器安全运维、网络安全运维和数据安全运维。

第1章讲述了Linux服务器安全运维，主要从系统角度介绍了账号安全、远程访问安全、文件系统安全、系统软件安全等的应用与防范，同时介绍了两款系统安全检测软件，最后通过一个实际案例详细介绍了服务器遭受攻击后的处理过程。

第2章讲述了Linux下网络安全运维，主要介绍了iftop、ntop、iperf、nmap等几款常用的网络安全运维工具，通过对这些网络安全工具的介绍，可使读者迅速定位网络故障与防范网络攻击。

第3章介绍了运维过程中的数据安全策略，主要介绍了数据镜像工具DRBD和数据恢复工具extundelete。DRBD可以通过网络对数据进行实时备份，保证数据安全，而extundelete可以在误删除数据的时候进行数据恢复。

运维故障排查篇（第4章和第5章）

运维故障排查篇介绍了运维工作中可能遇到的一些常见问题以及相应的解决思路和方法。

第4章讲述了Linux服务器经常出现的问题及解决思路，这是Linux运维的基础。

第5章讲述了Linux运维中常见的一些软件级应用故障，以及故障出现的原因和最终的解决方法。该章内容完全是生产环境下的实际案例，通过对每个案例的介绍，更多传递的是一种解决问题的思路，相信掌握了思路，一切问题都会迎刃而解。

自动化运维篇（第6章至第9章）

自动化运维篇是本书的一个重点，主要讲述了海量主机的自动化部署、分布式监控等内容。

第6章讲述了几个轻量级自动化部署工具，分别是pssh、pdsh和mushsh，这三个小工具基本可以应对在上千台主机中批量安装和部署软件。

第7章讲述了分布式监控系统Ganglia的使用方法和应用实例，通过Ganglia可以监控千台以上Linux主机，并且性能稳定。

第8章讲述了如何通过nagios构建一个基于Web的分布式监控报警平台，其实就是将nagios监控、报警配置Web化的过程，而分布式Web监控平台是通过一款开源软件Centreon实现的，并可实现声音、邮件、短信等多种形式的报警。

第9章讲述了如何构建一个智能化的监控报警平台，主要讲述如何将Ganglia和Centreon实现整合。在这个整合过程中，Ganglia是一个数据收集平台，而Centreon是一个数据展示平台，通过一个数据提取程序实现了Ganglia和Centreon的无缝整合。

集群架构篇（第10章至第14章）

集群架构应用篇也是本书的重点，主要介绍了基于Web和数据库的高可用集群、负载均衡集群的应用案例，每个案例都是生产环境下的真实应用。

第 10 章讲述了 Nginx 作为 Web 服务器的应用, 主要介绍了 Nginx 的反向代理功能和 URL 重写功能, 并列举了很多实例, 最后深入讲述了 Nginx 作为 Web 缓存服务器、Nginx 作为负载均衡服务器的应用案例。

第 11 章讲述了高性能集群软件 Keepalived 的实战应用, 首先介绍了 Keepalived 的实现原理, 接着详细介绍了 Keepalived 的配置过程, 最后通过一个应用案例演示了 Keepalived 在生产环境下的使用方法和使用技巧。

第 12 章讲述了千万级高并发负载均衡软件 HAProxy 的应用技巧, 首先介绍了 HAProxy 常见的应用实例和配置技巧, 然后通过生产环境下 HAProxy 的配置实例详细介绍了基于虚拟主机的负载均衡的实现过程。

第 13 章讲述了如何构建高性能的 MySQL 集群系统, 主要介绍了常见的高可用 MySQL 解决方案, 深入讲述了通过 Keepalived 实现 MySQL 双主高可用、MMM 构建 MySQL 高可用、MySQL 读写分离等各种 MySQL 生产环境高性能解决方案。

第 14 章讲述了如何构建高可用的 HAProxy 负载均衡集群系统, 通过单机 HAProxy 的缺点, 引出构建高可用 HAProxy 的必要性, 详细介绍了通过 Keepalived 实现 HAProxy 高可用的解决方案。

勘误和支持

由于作者的水平有限, 加之编写时间仓促, 书中难免会出现一些错误或不准确的地方, 不妥之处恳请读者批评指正。

本书的修订信息会发布在笔者的博客上, 地址为 <http://ixdba.blog.51cto.com>。笔者会在该博客中不定期更新本书中的遗漏, 当然, 也欢迎读者将遇到疑惑或书中的错误在博客留言中提出。如果你有更多的宝贵意见, 也欢迎发送邮件至笔者的邮箱 (m13388@163.com), 期待能够听到你的真挚反馈。

致谢

首先要感谢我的爸爸、妈妈, 感谢你们将我培养成人, 并时时刻刻向我传递信心和力量!

感谢我的妻子吴娟然女士, 是她的鼓励和背后默默的支持, 让我坚持写完了这本书。

感谢对本书提供大力支持的杨武先生、禄广峰先生, 感谢我的挚友张建坤、兰海文, 他们从技术角度对本书某些章节进行了修改和补充, 并提出了很多意见和建议。

感谢机械工业出版社华章公司的编辑杨福川老师和姜影老师, 此书的出版离不开他们的辛苦付出。

本书内容是建立在开源软件与开源社区研究成果基础之上的，因此在本书完成之际，对无私奉献的每位开源作者以及每个开源社区表示衷心的感谢，因为有他们，开源世界才更加精彩。同时也要感谢学习和使用 Linux 开源软件过程中认识的一些同行好友，以及众多本书的支持者，在本书撰写过程中他们向我提出了很多意见和建议，人数众多不一列举，在此一并感谢。

本书定稿之时，我儿子已经满三个月了，在这里祝福我的宝贝健康成长、幸福快乐，谨以此书作为送给他的一份特殊的人生礼物！

高俊峰（南非蚂蚁）

2014 年 4 月于西安

目 录 Contents

前言

第一部分 安全运维篇

第 1 章 Linux 服务器安全运维	3
1.1 账户和登录安全	3
1.1.1 删除特殊的用户和用户组	3
1.1.2 关闭系统不需要的服务	4
1.1.3 密码安全策略	5
1.1.4 合理使用 su、sudo 命令	9
1.1.5 删减系统登录欢迎信息	11
1.1.6 禁止 Control-Alt-Delete 键盘关闭命令	12
1.2 远程访问和认证安全	12
1.2.1 远程登录取消 telnet 而采用 SSH 方式	12
1.2.2 合理使用 shell 历史命令记录功能	14
1.2.3 启用 tcp_wrappers 防火墙	16
1.3 文件系统安全	18
1.3.1 锁定系统重要文件	18
1.3.2 文件权限检查和修改	20
1.3.3 /tmp、/var/tmp、/dev/shm 安全设定	21
1.4 系统软件安全管理	22
1.4.1 软件自动升级工具 yum	23
1.4.2 yum 的安装与配置	23

1.4.3 yum 的特点与基本用法	25
1.4.4 几个不错的 yum 源	27
1.5 Linux 后门入侵检测工具	28
1.5.1 rootkit 后门检测工具 chkrootkit	29
1.5.2 rootkit 后门检测工具 RKHunter	31
1.6 服务器遭受攻击后的处理过程	35
1.6.1 处理服务器遭受攻击的一般思路	35
1.6.2 检查并锁定可疑用户	36
1.6.3 查看系统日志	37
1.6.4 检查并关闭系统可疑进程	37
1.6.5 检查文件系统的完好性	38
1.7 一次 Linux 被入侵后的分析	39
1.7.1 受攻击现象	39
1.7.2 初步分析	40
1.7.3 断网分析系统	40
1.7.4 寻找攻击源	41
1.7.5 查找攻击原因	42
1.7.6 揭开谜团	43
1.7.7 如何恢复网站	43
第 2 章 Linux 网络安全运维	45
2.1 网络实时流量监测工具 iftop	45
2.1.1 iftop 能做什么	45
2.1.2 iftop 的安装	45
2.1.3 使用 iftop 监控网卡实时流量	46
2.2 网络流量监控与分析工具 Ntop 和 Ntopng	49
2.2.1 Ntop 与 MRTG 的异同	49
2.2.2 Ntop 与 Ntopng 的功能介绍	49
2.2.3 安装 Ntop 与 Ntopng	50
2.2.4 Ntop 和 Ntopng 的使用技巧	53
2.3 网络性能评估工具 iperf	60
2.3.1 iperf 能做什么	60
2.3.2 iperf 的安装与使用	61

2.3.3	iperf 应用实例	62
2.4	网络探测和安全审核工具 nmap	67
2.4.1	nmap 和 Zenmap 简介	67
2.4.2	nmap 基本功能与结构	67
2.4.3	nmap 的安装与验证	68
2.4.4	nmap 的典型用法	68
2.4.5	nmap 主机发现扫描	70
2.4.6	nmap 端口扫描	71
2.4.7	nmap 版本探测	73
2.4.8	nmap 操作系统探测	74
第 3 章	数据安全工具 DRBD、extundelete	77
3.1	数据镜像软件 DRBD 介绍	77
3.1.1	DRBD 的基本功能	77
3.1.2	DRBD 的构成	78
3.1.3	DRBD 与现在的集群的关系	78
3.1.4	DRBD 的主要特性	79
3.2	DRBD 的安装与配置	80
3.2.1	安装环境说明	80
3.2.2	DRBD 的安装部署	81
3.2.3	快速配置一个 DRBD 镜像系统	82
3.3	DRBD 的管理与维护	83
3.3.1	启动 DRBD	83
3.3.2	测试 DRBD 数据镜像	85
3.3.3	DRBD 主备节点切换	86
3.4	数据恢复软件 extundelete 介绍	88
3.4.1	如何使用“rm -rf”命令	88
3.4.2	extundelete 与 ext3grep 的异同	89
3.4.3	extundelete 的恢复原理	89
3.4.4	安装 extundelete	89
3.4.5	extundelete 用法详解	90
3.5	实战：extundelete 恢复数据的过程	91
3.5.1	通过 extundelete 恢复单个文件	91

3.5.2 通过 extundelete 恢复单个目录	93
3.5.3 通过 extundelete 恢复所有误删除数据	93
3.5.4 通过 extundelete 恢复某个时间段的数据	94

第二部分 运维故障排查篇

第4章 Linux 系统运维故障排查思路	97
----------------------------	----

4.1 Linux 系统故障的处理思路	97
4.2 Linux 系统无法启动的解决方法	98
4.2.1 文件系统破坏导致系统无法启动	98
4.2.2 /etc/fstab 文件丢失导致系统无法启动	100
4.3 Linux 系统无响应(死机)问题分析	104
4.4 Linux 下常见网络故障的处理思路	105
4.4.1 检查网络硬件问题	105
4.4.2 检查网卡是否正常工作	105
4.4.3 检查 DNS 解析文件是否设置正确	106
4.4.4 检查服务是否正常打开	107
4.4.5 检查访问权限是否打开	108
4.4.6 检查局域网主机之间联机是否正常	109

第5章 Linux 故障排查案例实战	111
--------------------------	-----

5.1 常见系统故障案例	111
5.1.1 su 切换用户带来的疑惑	111
5.1.2 “Read-only file system” 错误与解决方法	114
5.1.3 “Argument list too long” 错误与解决方法	116
5.1.4 inode 耗尽导致应用故障	119
5.1.5 文件已删除但空间不释放的原因	121
5.1.6 “Too many open files” 错误与解决方法	124
5.2 Apache 常见错误故障案例	127
5.2.1 “No space left on device” 错误与解决方法	127
5.2.2 apache(20014) 故障与解决方法	129
5.2.3 “could not bind to address 0.0.0.0:80” 错误与解决方法	131

5.3 因 NAS 存储故障引起的 Linux 系统恢复案例	134
5.3.1 故障现象描述	134
5.3.2 问题判断思路	134
5.3.3 问题处理过程	134
5.3.4 解决问题	137

第三部分 自动化运维篇

第 6 章 轻量级运维利器 pssh、pdsh 和 mussh	141
---------------------------------	-----

6.1 并行 SSH 运维工具 pssh	141
6.1.1 pssh 应用场景	141
6.1.2 pssh 的安装与用法	142
6.1.3 pssh 应用实例	144
6.2 并行分布式运维工具 pdsh	147
6.2.1 pdsh 应用场景	147
6.2.2 pdsh 的安装与语法	148
6.2.3 pdsh 应用实例	149
6.3 多主机 ssh 封装器 mussh	153
6.3.1 mussh 功能介绍	153
6.3.2 mussh 的安装与语法	153
6.3.3 mussh 应用实例	154

第 7 章 分布式监控系统 Ganglia	157
-----------------------	-----

7.1 Ganglia 简介	157
7.2 Ganglia 的组成	157
7.3 Ganglia 的工作原理	159
7.3.1 Ganglia 数据流向分析	159
7.3.2 Ganglia 工作模式	160
7.4 Ganglia 的安装	161
7.4.1 yum 源安装方式	161
7.4.2 源码方式	162
7.5 配置一个 Ganglia 分布式监控系统	164

7.5.1	Ganglia 配置文件介绍	164
7.5.2	Ganglia 监控系统架构图	164
7.5.3	Ganglia 监控管理端配置	164
7.5.4	Ganglia 的客户端配置	165
7.5.5	Ganglia 的 Web 端配置	166
7.6	Ganglia 监控系统的管理和维护	167
7.7	Ganglia 监控扩展实现机制	169
7.7.1	扩展 Ganglia 监控功能的方法	169
7.7.2	通过 gmetric 接口扩展 Ganglia 监控	169
7.7.3	通过 Python 插件扩展 Ganglia 监控	170
7.7.4	实战：利用 Python 接口监控 Nginx 运行状态	171
7.8	Ganglia 在实际应用中要考虑的问题	174
7.8.1	网络 IO 可能存在瓶颈	174
7.8.2	CPU 可能存在瓶颈	175
7.8.3	gmetad 写入 rrd 数据库可能存在瓶颈	175
第 8 章	基于 nagios 的分布式监控报警平台 Centreon	177
8.1	Centreon 概述	177
8.2	Centreon 的特点	177
8.3	Centreon 的结构	178
8.4	安装 Centreon+nagios 监控系统	179
8.4.1	安装支持 Centreon 的 yum 源	180
8.4.2	安装系统基础依赖库	180
8.4.3	安装 nagios 及 nagios-plugins	181
8.4.4	安装 ndoutils	181
8.4.5	安装 nrpe	182
8.4.6	安装 Centreon	182
8.4.7	安装配置 Centreon Web	191
8.4.8	启动 Centreon 相关服务	195
8.4.9	安装问题总结	196
8.5	配置 Centreon 监控系统	197
8.5.1	添加主机和主机组	197
8.5.2	批量添加主机	202

8.5.3	监控引擎管理	206
8.5.4	添加服务和 service 组	207
8.5.5	监控报警配置	211
8.5.6	用户和用户权限管理	217
8.6	配置分布式监控	222
8.6.1	分布式监控架构与实现原理	222
8.6.2	分布式监控搭建环境介绍	223
8.6.3	监控软件的安装	224
8.6.4	配置节点间 SSH 信任登录	224
8.6.5	在 Central server 上添加分布式监控配置	226
8.7	常见服务监控配置	230
8.7.1	nagios 插件编写规范	231
8.7.2	监控 Apache 运行状态	231
8.7.3	监控 MySQL 运行状态	234
8.7.4	监控 Hadoop HDFS 运行状态	237
8.8	桌面监控报警器 Nagstamon	239
第 9 章	通过 Ganglia 与 Centreon 构建智能化监控报警平台	243
9.1	智能运维监控报警平台的组成	243
9.2	Ganglia 作为数据收集模块	246
9.3	Centreon 作为监控报警模块	246
9.4	Ganglia 与 Centreon 的无缝整合	247
9.4.1	数据提取脚本	247
9.4.2	实现 Ganglia 与 Centreon 完美整合	256
9.5	在 Centreon 中实现批量数据收集与监控报警	259

第四部分 集群架构篇

第 10 章	高性能 Web 服务器 Nginx	267
10.1	高性能 Web 服务器 Nginx 介绍	267
10.1.1	Nginx 的组成与工作原理	267
10.1.2	Nginx 的性能优势	268

10.2	Nginx 的安装	269
10.2.1	安装 Nginx 依赖库	269
10.2.2	快速安装 Nginx	270
10.3	配置与调试 Nginx	270
10.3.1	Nginx 配置文件结构	270
10.3.2	Nginx 配置文件详解	270
10.3.3	Nginx 日常维护技巧	276
10.4	Nginx 常用功能介绍	278
10.4.1	Nginx 反向代理应用实例	278
10.4.2	Nginx 的 URL 重写应用实例	282
10.5	案例: Nginx 作为 Web 缓存服务器应用	286
10.5.1	在 Nginx 下安装缓存服务器	286
10.5.2	配置 Nginx 缓存服务器	287
10.5.3	测试 proxy_cache 实现的缓存功能	289
10.5.4	如何清除指定的 URL 缓存	290
10.6	案例: Nginx 作为负载均衡服务器应用	290
10.6.1	Nginx 的负载均衡算法	291
10.6.2	Nginx 的负载均衡配置实例	291
10.7	Nginx 性能优化技巧	292
10.7.1	编译安装过程优化	292
10.7.2	利用 TCMalloc 优化 Nginx 的性能	293
10.7.3	Nginx 内核参数优化	295
第 11 章	高性能集群软件 Keepalived	297
11.1	Keepalived 介绍	297
11.1.1	Keepalived 是什么	297
11.1.2	VRRP 协议与工作原理	298
11.1.3	Keepalived 工作原理	298
11.1.4	Keepalived 的体系结构	299
11.2	Keepalived 安装与配置	301
11.2.1	Keepalived 的安装过程	301
11.2.2	Keepalived 的全局配置	303
11.2.3	Keepalived 的 VRRPD 配置	303

11.2.4	Keepalived 的 LVS 配置	307
11.3	Keepalived 基础功能应用实例	310
11.3.1	Keepalived 基础 HA 功能演示	310
11.3.2	通过 vrrp_script 实现对集群资源的监控	316
11.3.3	Keepalived 集群中 MASTER 和 BACKUP 角色选举策略	319
第 12 章	千万级高并发负载均衡软件 HAProxy	323
12.1	高性能负载均衡软件 HAProxy 介绍	323
12.1.1	HAProxy 简介	323
12.1.2	四层和七层负载均衡的区别	324
12.1.3	HAProxy 与 LVS 的异同	325
12.2	HAProxy 基础配置与应用实例	326
12.2.1	快速安装 HAProxy 集群软件	326
12.2.2	HAProxy 基础配置文件详解	326
12.2.3	HAProxy 的日志配置策略	333
12.2.4	通过 HAProxy 的 ACL 规则实现智能负载均衡	334
12.3	基于虚拟主机的 HAProxy 负载均衡系统配置实例	335
12.3.1	通过 HAProxy 的 ACL 规则配置虚拟主机	335
12.3.2	测试 HAProxy 实现虚拟主机和负载均衡功能	343
12.3.3	测试 HAProxy 的故障转移功能	343
12.3.4	使用 HAProxy 的 Web 监控平台	343
第 13 章	构建高性能的 MySQL 集群系统	345
13.1	常见的高可用 MySQL 解决方案	345
13.1.1	主从复制解决方案	345
13.1.2	MMM 高可用解决方案	346
13.1.3	Heartbeat/SAN 高可用解决方案	346
13.1.4	Heartbeat/DRBD 高可用解决方案	346
13.1.5	MySQL Cluster 高可用解决方案	347
13.2	通过 Keepalived 搭建 MySQL 双主模式的高可用集群系统	347
13.2.1	MySQL Replication 介绍	347
13.2.2	MySQL Replication 实现原理	348
13.2.3	MySQL Replication 常用架构	349

13.2.4	MySQL 主主互备模式架构	349
13.2.5	MySQL 主主互备模式配置	350
13.2.6	配置 Keepalived 实现 MySQL 双主高可用	353
13.2.7	测试 MySQL 主从同步功能	356
13.2.8	测试 Keepalived 实现 MySQL 故障转移	358
13.3	通过 MMM 构建 MySQL 高可用集群系统	360
13.3.1	MMM 高可用 MySQL 方案简介	360
13.3.2	MMM 典型应用方案	361
13.3.3	MMM 高可用 MySQL 方案架构	363
13.3.4	MMM 的安装与配置	364
13.3.5	MMM 的管理	368
13.3.6	测试 MMM 实现 MySQL 高可用功能	371
13.4	MySQL 读写分离解决方案	374
13.4.1	通过 Amoeba 实现 MySQL 读写分离	374
13.4.2	通过 Keepalived 构建高可用的 Amoeba 服务	382
第 14 章	高性能负载均衡集群软件 HAProxy	383
14.1	高性能负载均衡架构设计原则	383
14.1.1	HAProxy 常见方案与拓扑	384
14.1.2	高可用集群软件的选择	386
14.2	搭建 HAProxy+Keepalived 高可用负载均衡系统	386
14.2.1	搭建环境描述	386
14.2.2	配置 HAProxy 负载均衡服务器	387
14.2.3	配置主、备 Keepalived 服务器	389
14.3	测试 HAProxy+Keepalived 高可用负载均衡集群	392
14.3.1	测试 Keepalived 的高可用功能	392
14.3.2	测试负载均衡功能	394
14.4	构建双主高可用的 HAProxy 负载均衡系统	394
14.4.1	系统架构图与实现原理	394
14.4.2	安装并配置 HAProxy 集群系统	395
14.4.3	安装并配置双主的 Keepalived 高可用系统	397
14.4.4	测试双主高可用的 HAProxy 负载均衡集群系统	399

第1章 Linux服务器安全运维

第1章 Chapter 1

Linux 服务器安全运维

第一部分 *Part 1*

安全运维篇

1.1 账户和登录安全

安全是IT行业一个老生常谈的话题了，最近时“棱镜门”事件，射出了很多安全问题，处理好信息安全问题已变得刻不容缓。因此作为一名运维人员，必须了解一些安全运维规则。同时，要保护自己所负责的业务，首先必须站在攻击者的角度思考问题，才能弥补任何潜在的威胁和漏洞。

账户安全是系统安全的第一道屏障，也是系统安全的核心。保障登录账户的安全，在一定程度上可以提高服务器的安全级别。本节重点介绍Linux系统登录账户的安全设置方法。

1.1.1 删除特殊的用户和用户组

- 第1章 Linux服务器安全运维
- 第2章 Linux网络安全运维
- 第3章 数据安全工具DRBD、extundelete

Linux 提供了各种不同用途的用户和用户组，如果不小心安装完成后，默认会安装很多不必要的用户和用户组，如果不及时删除，系统就会变得不安全。因为账户越多，系统就越不安全，从而很可能会导致系统被攻击。

Linux系统中可以删除的默认用户和用户组大致如下：

可删除的用户，如adm、lp、sync、shutdown、halt、news、uucp、operator、games、gopher等。

可删除的用户组，如adm、lp、news、uucp、games、dip、pppusers、popusers、slipusers等。

删除的方法很简单，下面以删除games用户和用户组为例介绍具体的操作。

删除系统不必要的用户使用下面命令：

Linux 服务器安全运维

1.1 账户和登录安全

安全是 IT 行业一个老生常谈的话题了，最近的“棱镜门”事件折射出了很多安全问题，处理好信息安全问题已变得刻不容缓。因此作为一名运维人员，必须了解一些安全运维准则，同时，要保护自己所负责的业务，首先要站在攻击者的角度思考问题，才能修补任何潜在的威胁和漏洞。

账户安全是系统安全的第一道屏障，也是系统的核心，保障登录账户的安全，在一定程度上可以提高服务器的安全级别，本节重点介绍 Linux 系统登录账户的安全设置方法。

1.1.1 删除特殊的用户和用户组

Linux 提供了各种不同角色的系统账号，在系统安装完成后，默认会安装很多不必要的用户和用户组，如果不需要某些用户或者用户组，应立即删除它们，因为账户越多，系统就越不安全，从而很可能被黑客利用，威胁服务器的安全。

Linux 系统中可以删除的默认用户和用户组大致如下：

□ 可删除的用户，如 adm、lp、sync、shutdown、halt、news、uucp、operator、games、gopher 等。

□ 可删除的用户组，如 adm、lp、news、uucp、games、dip、pppusers、popusers、slipusers 等。

删除的方法很简单，下面以删除 games 用户和用户组为例介绍具体的操作。

删除系统不必要的用户使用下面命令：


```
[root@localhost ~]# userdel games
```

删除系统不必要的用户组使用如下命令：

```
[root@localhost ~]# groupdel games
```

有些时候，某些用户仅仅用作进程调用或者用户组调用，并不需要登录功能，此时可以禁止这些用户登录系统的功能，例如要禁止 nagios 用户的登录功能，可以执行如下命令：

```
[root@localhost ~]# usermod -s /sbin/nologin nagios
```

其实要删除哪些用户和用户组，并没有固定要求，可以根据服务器的用途来决定，如果服务器是用于 Web 应用的，那么系统默认的 apache 用户和用户组就无需删除；而如果服务器是用于数据库应用的，那么建议删除系统默认的 apache 用户和用户组。

1.1.2 关闭系统不需要的服务

在安装完成后，Linux 绑定了很多没用的服务，这些服务默认都是自动启动的。对于服务器来说，运行的服务越多，系统就越不安全，运行的服务越少，系统安全性就越高，因此关闭一些不需要的服务，对系统安全有很大的帮助。

具体关闭哪些服务，要根据服务器的用途而定，一般情况下，只要系统本身用不到的服务都认为是不必要的服务，例如某台 Linux 服务器用于 WWW 应用，那么除了 httpd 服务和系统运行是必需的服务外，其他服务都可以关闭。下面这些服务一般情况下是不需要的，可以选择关闭：

anacron、auditd、autofs、avahi-daemon、avahi-dnscnf、bluetooth、cpuspeed、firstboot、gpm、haldaemon、hidd、ip6tables、ipsec、isdn、lpd、mcstrans、messagebus、netfs、nfs、nfslock、nscd、pcscd portmap、readahead_early、restorecond、rpcgssd、rpcidmapd、rstatd、sendmail、setroubleshoot、ypasswdd ypserv

关闭服务自动启动的方法很简单，可以通过 chkconfig 命令实现。例如，要关闭 bluetooth 服务，执行下面命令即可：

```
chkconfig --level 345 bluetooth off
```

对所有需要关闭的服务都执行上面的操作后，重启服务器即可。

为了系统能够正常稳定运行，建议启动的系统运行必需的服务如表 1-1 所示。

表 1-1 系统运行必需的服务

服务名称	服务内容
acpid	用于电源管理，对于笔记本电脑和台式电脑很重要，建议开启
apmd	高级电源能源管理服务，可以监控电池性能
kudzu	检测硬件是否变化的服务，建议开启

(续)

服务名称	服务内容
cron	为 Linux 下自动安排的进程提供运行服务，建议开启
atd	类似 cron，提供在指定的时间做指定的事情的服务，与 Windows 下的计划任务功能相同
keytable	用于装载镜像键盘。根据情况，可以选择启动
iptables	Linux 内置的防火墙软件，为了系统安全，必须启动
xinetd	支持多种网络服务的核心守候进程，建议开启
xfs	使用 X Window 桌面系统必需的服务
network	激活已配置网络接口的脚本程序，也就是启动网络服务，当然要启动
sshd	提供远程登录 Linux 上的服务，为了系统维护方便，一般建议开启
syslog	记录系统日志的服务，很重要，建议开启

1.1.3 密码安全策略

在 Linux 下，远程登录系统有两种认证方式：密码认证和密钥认证。密码认证方式是传统的安全策略，对于密码的设置，比较普遍的说法是：至少 6 个字符以上，密码要包含数字、字母、下划线、特殊符号等。设置一个相对复杂的密码，对系统安全能起到一定的防护作用，但是也面临一些其他问题，例如密码暴力破解、密码泄露、密码丢失等，同时过于复杂的密码也会对运维工作造成一定的负担。

密钥认证是一种新型的认证方式，公用密钥存储在远程服务器上，专用密钥保存在本地，当需要登录系统时，通过本地专用密钥和远程服务器的公用密钥进行配对认证，如果认证成功，就可以成功登录系统。这种认证方式避免了被暴力破解的危险，同时只要保存在本地的专用密钥不被黑客盗用，攻击者一般无法通过密钥认证的方式进入系统。因此，在 Linux 下推荐用密钥认证方式登录系统，这样就可以抛弃密码认证登录系统的弊端。

Linux 服务器一般通过 SecureCRT、putty、Xshell 之类的工具进行远程维护和管理，密钥认证方式的实现就是借助于 SecureCRT 软件和 Linux 系统中的 SSH 服务实现的。

SSH 的英文全称是 Secure SHell。SSH 以及 OpenSSH 是类似于 telnet 的远程登录程序，SecureCRT 就是一个 SSH 客户端，要想通过 SecureCRT 登录远程机器，要求该远程机器必须运行 sshd 服务。但是，与 telnet 不同的是，SSH 协议非常安全，数据流加密传输，确保数据流的完整性和安全性。OpenSSH 的 RSA/DSA 密钥认证系统是一个优秀的功能组件，使用基于密钥认证系统的优点在于：在许多情况下，可以不必手工输入密码就能建立起安全的连接。

支持 RSA/DSA 密钥认证的软件有很多，这里以 SecureCRT 为例，详细讲述通过密钥认证方式远程登录 Linux 服务器的实现方法。

这里的环境是 SecureCRT5.1、CentOS6.4、SSH-2.0-OpenSSH_5.3，操作如下。

1) 首先产生 SSH2 的密钥对, 这里选择使用 RSA 1024 位加密, 如图 1-1 所示。

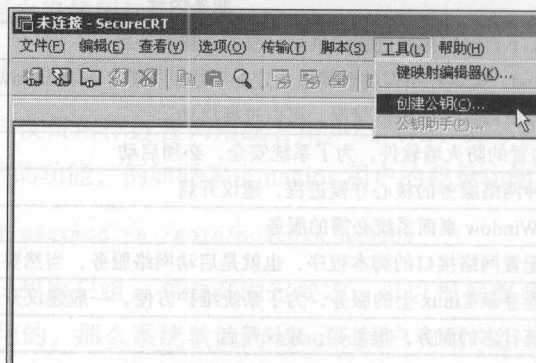


图 1-1 创建公钥

2) 弹出“密钥生成向导”对话框, 如图 1-2 所示。

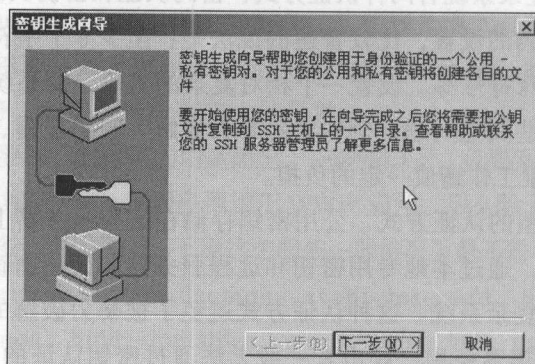


图 1-2 密钥生成向导

3) 在选择密钥类型时, 选择 RSA 方式, 如图 1-3 所示。

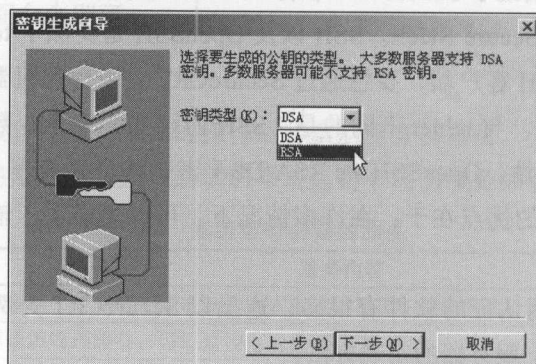


图 1-3 选择密钥类型

4) 输入一个保护设定的加密密钥的通行短语, 如图 1-4 所示。

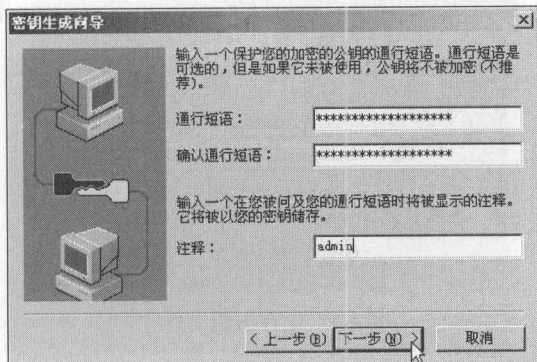


图 1-4 设定加密密钥通行短语

5) 在密钥的长度(位)中, 使用默认的 1024 位加密即可, 如图 1-5 所示。

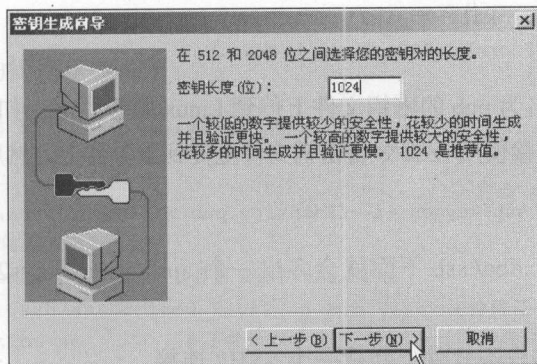


图 1-5 设置密钥长度

6) 系统开始生成密钥, 如图 1-6 所示。

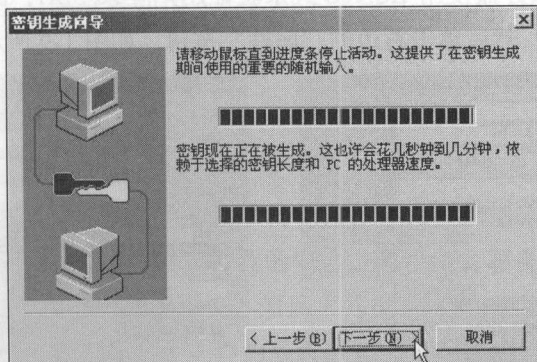


图 1-6 生成密钥

7) 为生成的密钥选择一个文件名和存放的目录(可以自行修改或使用默认值),如图 1-7 所示。

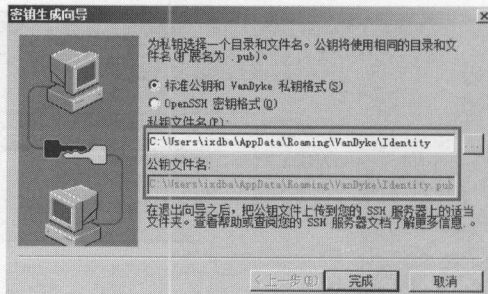


图 1-7 保存密钥文件

到这里为止,使用客户端 SecureCRT 生成密钥的步骤已经完成。接下来将密钥文件上传到 Linux 服务器端,并在服务器端导入密钥。

例如,这里设置普通用户 ixdba 使用 SSH2 协议,在 Linux 服务器执行如下操作:

```
[ixdba@localhost~]$ mkdir /home/ixdba/.ssh
[ixdba@localhost~]$ chmod 700 /home/ixdba/.ssh
```

把之前生成的后缀名为 pub 的密钥文件上传到 Linux 服务器上,如果已经在用 SecureCRT 连接 Linux 系统,可以直接使用 rz 命令将密钥文件上传到服务器上,然后开始导入:

```
[ixdba@localhost~]$ ssh-keygen -i -fIdentity.pub >> /home/ixdba/.ssh/authorized_keys2
```

完成后,在 /home/ixdba/.ssh 下面就会多出一个 authorized_keys2 文件,这就是服务器端的密钥文件。

8) 在 SecureCRT 客户端软件上新建一个 SSH2 连接。

在“协议”下拉列表中选择“SSH2”,在主机名文本框中输入“192.168.12.188”,在用户名文本框中输入“ixdba”,其他保持默认,如图 1-8 所示。

9) 由于这里要让服务器使用 RSA 方式来验证用户登录 SSH,因此在“鉴权”一栏中只需选择“公钥”方式,然后单击右边的“属性”按钮,如图 1-9 所示。

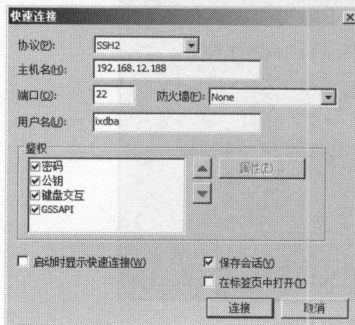


图 1-8 新建一个 SSH2 连接

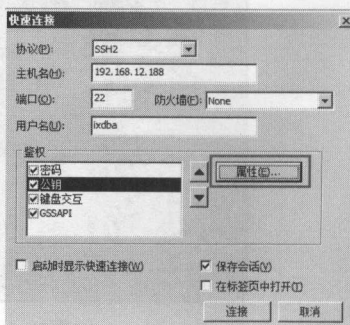


图 1-9 指定身份验证方式

10) 在出现的“公钥属性”窗口中选择“使用身份或证书文件”，然后找到上面步骤 7 中生成的密钥文件，如图 1-10 所示。

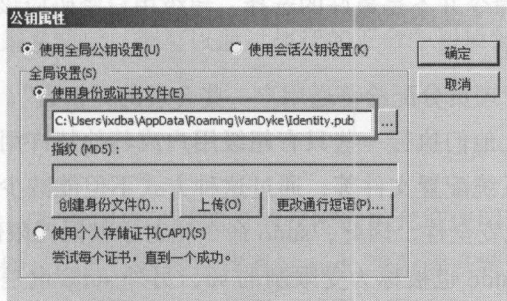


图 1-10 指定身份或证书文件

11) 到此为止，通过 RSA 密钥方式验证用户登录 SSH 的步骤就全部完成了。接下来，为了服务器的安全，还需要修改 SSH2 的配置文件，让其只能接收 PublicKey 认证方式来验证用户。

在 Linux 服务器上的操作步骤如下：

```
[root@localhost ~]# vi /etc/ssh/sshd_config
```

修改如下几个配置：

```
Protocol 2 # 仅允许使用 SSH2
PubkeyAuthentication yes # 启用 PublicKey 认证
AuthorizedKeysFile .ssh/authorized_keys2 # PublicKey 文件路径
PasswordAuthentication no # 不使用口令认证
```

最后重启 sshd 服务，执行如下命令：

```
[root@localhost ~]# /etc/rc.d/init.d/sshd restart
```

等 sshd 服务启动完毕，就可以利用 SecureCRT 通过 PublicKey 认证远程登录 Linux 系统了。

1.1.4 合理使用 su、sudo 命令

su 命令是一个切换用户的工具，经常用于将普通用户切换到超级用户下，当然也可以从超级用户切换到普通用户。为了保证服务器的安全，几乎所有服务器都禁止了超级用户直接登录系统，而是通过普通用户登录系统，然后再通过 su 命令切换到超级用户下，执行一些需要超级权限的工作。通过 su 命令能够为系统管理带来一定的方便，但是也存在不安全的因素，例如系统有 10 个普通用户，每个用户都需要执行一些有超级权限的操作，就必

须把超级用户的密码交给这 10 个普通用户，如果这 10 个用户都有超级权限，通过超级权限可以做什么事，那么在一定程度上会对系统的安全造成威胁。因此在很多人都需要参与的系统管理中，使用 su 命令并不是最好的选择，超级用户密码应该掌握在少数人手中，此时 sudo 命令就派上用场了。

sudo 命令允许系统管理员分配给普通用户一些合理的“权力”，并且不需要普通用户知道超级用户密码，就能让他们执行一些只有超级用户或其他特许用户才能完成的任务，比如系统服务重启、编辑系统配置文件等，通过这种方式不但能减少超级用户登录次数和管理时间，而且提高了系统安全性。因此，sudo 命令相对于权限无限制性的 su 命令来说，还是比较安全的，这使得 sudo 也被称为受限制的 su，另外 sudo 也是需要事先进行授权认证的，所以也被称为授权认证的 su。

sudo 执行命令的流程是：将当前用户切换到超级用户下，或切换到指定的用户下，然后以超级用户或其指定切换到的用户身份执行命令，执行完成后，直接退回到当前用户，而这一切的完成要通过 sudo 的配置文件 /etc/sudoers 来进行授权。

例如，普通用户是无法访问 /etc/shadow 文件的：

```
[user01@unknown ~]$ more /etc/shadow
/etc/shadow: Permission denied
```

如果要让普通用户 user01 可访问这个文件，可以在 /etc/sudoers 添加如下内容：

```
user01    ALL = /bin/more /etc/shadow
```

这样，通过如下方式 user01 用户就可访问 /etc/shadow 文件：

```
[user01@unknown ~]$ sudo more /etc/shadow
[sudo] password for user01:
```

执行这个命令后，需要输入 user01 用户的密码，然后就可访问文件内容了。在这里 sudo 使用时间戳文件来完成类似“检票”的系统功能，当用户输入密码后就获得了一张默认存活期为 5 分钟的“入场券”（默认值可以在编译的时候改变）。超时以后，用户必须重新输入密码才能查看文件内容。

如果每次都需要输入密码，那么某些自动调用超级权限的程序就会出现问題，此时可以通过下面的设置让普通用户无需输入密码即可执行具有超级权限的程序。例如，要让普通用户 centreon 具有 /etc/init.d/nagios 脚本重启的权限，可以在 /etc/sudoers 添加如下设置：

```
CENTREON  ALL = NOPASSWD: /etc/init.d/nagios restart
```

这样，普通用户 centreon 就可以执行重启 nagios 脚本而无需输入密码了。如果要让一个普通用户 user02 具有超级用户的所有权限，而又不想输入超级用户的密码，只需在 /etc/

sudoers 添加如下内容即可：

```
user02 ALL=(ALL) NOPASSWD: ALL
```

这样 user02 用户登录系统后，就可以通过执行如下命令切换到超级用户：

```
[user02@unknown ~]$ sudo su -  
[root@unknown ~]# pwd  
/root
```

设计 sudo 的宗旨是：赋予用户尽可能少的权限但仍允许他们完成自己的工作，这种设计兼顾了安全性和易用性，因此，强烈推荐通过 sudo 来管理系统账号的安全，只允许普通用户登录系统，如果这些用户需要特殊的权限，就通过配置 /etc/sudoers 来完成，这也是多用户系统下账号安全管理的基本方式。

1.1.5 删减系统登录欢迎信息

虽然系统的一些欢迎信息或版本信息能给系统管理者带来一定的方便，但是这些信息有时候可能被黑客利用，从而成为攻击服务器的帮凶，为了保证系统的安全，可以修改或删除某些系统文件，这样的文件有 4 个，分别是 /etc/issue、/etc/issue.net、/etc/redhat-release 和 /etc/motd。

/etc/issue 和 /etc/issue.net 文件都记录了操作系统的名称和版本号，当用户通过本地终端或本地虚拟控制台等登录系统时，就会显示 /etc/issue 的文件内容，当用户通过 ssh 或 telnet 等远程登录系统时，在登录后就会显示 /etc/issue.net 的文件内容。在默认情况下 /etc/issue.net 文件的内容是不会在 ssh 登录后显示的，要显示这个信息可以修改 /etc/ssh/sshd_config 文件，在此文件中添加如下内容即可：

```
Banner /etc/issue.net
```

其实，这些登录提示很明显泄露了系统信息，为了安全起见，建议将此文件中的内容删除或修改。

/etc/redhat-release 文件也记录了操作系统的名称和版本号，为了安全起见，可以将此文件中的内容删除。

/etc/motd 文件是系统的公告信息。每次用户登录后，/etc/motd 文件的内容就会显示在用户的终端。通过这个文件系统，管理员可以发布一些软件或硬件的升级、系统维护等通告信息，但是此文件的最大作用就是可以发布一些警告信息，在黑客登录系统后，会对其发出这些警告信息，进而产生一些震慑作用。笔者曾看过国外的一个报道，黑客入侵了一台服务器，而这台服务器却给出了欢迎登录的信息，因此法院不做任何裁决。

1.1.6 禁止 Control-Alt-Delete 键盘关闭命令

在 Linux 的默认设置下，同时按下 Control-Alt-Delete 组合键，系统将自动重启，这个策略是很不安全的，因此要禁止 Control-Alt-Delete 组合键重启系统。禁止的方法很简单，在 CentOS5.x 以下的系统，只需修改 /etc/inittab 文件即可，操作如下：

```
[root@localhost ~]# vi /etc/inittab
```

找到如下这行：

```
ca::ctrlaltdel:/sbin/shutdown -t3 -r now
```

在这行之前加上“#”，然后执行：

```
[root@localhost ~]# telinit q
```

在 CentOS6.x 以上版本中，需要修改 /etc/init/control-alt-delete.conf 文件，找到如下内容：

```
exec /sbin/shutdown -r now "Control-Alt-Delete pressed"
```

在这行之前加上“#”进行注释掉即可。

1.2 远程访问和认证安全

1.2.1 远程登录取消 telnet 而采用 SSH 方式

telnet 是一种古老的远程登录认证服务，它在网络上用明文传送口令和数据，因此别有用心的人就会非常容易截获这些口令和数据。而且，telnet 服务程序的安全验证方式也极其脆弱，攻击者可以轻松将虚假信息传送给服务器。现在远程登录基本抛弃了 telnet 这种方式，取而代之的是通过 SSH 服务远程登录服务器。

关于 SSH 在前面已经做过一些简单的介绍，它是由客户端和服务端端的软件组成的，在客户端可以使用的软件有 SecureCRT、putty、Xshell 等，而在服务器端运行的是一个 sshd 服务。通过使用 SSH，可以加密所有传输的数据，而且能够防止 DNS 和 IP 欺骗。使用 SSH 的另外一个好处就是：传输的数据是经过压缩的，所以可以加快网络传输速度。

下面重点介绍下如何配置服务器端的 sshd 服务，以保证服务器远程连接的安全。

sshd 服务对应的主配置文件是 /etc/ssh/sshd_config，下面重点介绍下此文件关于安全方面的几个配置。先打开主配置文件：

```
[root@localhost ~]# vi /etc/ssh/sshd_config
```

主配置文件中各个配置选项的含义如下：

- ❑ Port 22, “Port” 用来设置 sshd 监听的端口, 为了安全起见, 建议更改默认的 22 端口, 选择 5 位以上的陌生数字端口。
- ❑ Protocol 2, 设置使用的 SSH 协议的版本为 SSH1 或 SSH2, SSH1 版本有缺陷和漏洞, 因此这里选择 Protocol 2 即可。
- ❑ ListenAddress 0.0.0.0, “ListenAddress” 用来设置 sshd 服务器绑定的 IP 地址。
- ❑ HostKey /etc/ssh/ssh_host_dsa_key, “HostKey” 用来设置服务器密匙文件的路径。
- ❑ KeyRegenerationInterval 1h, “KeyRegenerationInterval” 用来设置在多少秒之后系统自动重新生成服务器的密匙 (如果使用密匙)。重新生成密匙是为了防止利用盗用的密匙解密被截获的信息)。
- ❑ ServerKeyBits 1024, “ServerKeyBits” 用来定义服务器密匙的长度。
- ❑ SyslogFacility AUTHPRIV, “SyslogFacility” 用来设定在记录来自 sshd 的消息的时候, 是否给出 “facility code”。
- ❑ LogLevel INFO, “LogLevel” 用来记录 sshd 日志消息的级别。
- ❑ LoginGraceTime 2m, “LoginGraceTime” 用来设置如果用户登录失败, 在切断连接前服务器需要等待的时间, 以秒为单位。
- ❑ PermitRootLogin no, “PermitRootLogin” 用来设置超级用户 root 能不能用 SSH 登录。root 远程登录 Linux 是很危险的, 因此在远程 SSH 登录 Linux 系统时, 建议将这个选项设置为 “no”。
- ❑ StrictModes yes, “StrictModes” 用来设置 SSH 在接收登录请求之前是否检查用户根目录和 rhosts 文件的权限和所有权。建议将此选项设置为 “yes”。
- ❑ RSAAuthentication no, “RSAAuthentication” 用来设置是否开启 RSA 密钥验证, 只针对 SSH1, 如果采用 RSA 密钥登录方式时, 开启此选项。
- ❑ PubkeyAuthentication yes, “PubkeyAuthentication” 用来设置是否开启公钥验证, 如果采用公钥验证方式登录时, 开启此选项。
- ❑ AuthorizedKeysFile .ssh/authorized_keys, “AuthorizedKeysFile” 用来设置公钥验证文件的路径, 与 PubkeyAuthentication 配合使用。
- ❑ IgnoreUserKnownHosts no, “IgnoreUserKnownHosts” 用来设置 SSH 在进行 RhostsRSAAuthentication 安全验证时是否忽略用户的 “\$HOME/.ssh/known_hosts” 文件。
- ❑ IgnoreRhosts yes, “IgnoreRhosts” 用来设置验证的时候是否使用 “~/.rhosts” 和 “~/.shosts” 文件。
- ❑ PasswordAuthentication yes, “PasswordAuthentication” 用来设置是否开启密码验证

机制，如果使用密码登录系统，应该设置为“yes”。

- ☐ PermitEmptyPasswords no, “PermitEmptyPasswords”用来设置是否允许用口令为空的账号登录系统，必须选择“no”。
- ☐ ChallengeResponseAuthentication no, 禁用 s/key 密码。
- ☐ UsePAM no, 不通过 PAM 验证。
- ☐ X11Forwarding yes, “X11Forwarding”用来设置是否允许 X11 转发。
- ☐ PrintMotd yes, “PrintMotd”用来设置 sshd 是否在用户登录的时候显示“/etc/motd”中的信息，可以在 /etc/motd 中加入警告信息，以震慑攻击者。
- ☐ PrintLastLog no, 是否显示上次登录信息，设置为“no”表示不显示。
- ☐ Compression yes, 是否压缩命令，建议选择“yes”。
- ☐ TCPKeepAlive yes, 选择“yes”防止死连接。
- ☐ UseDNS no, 是否使用 DNS 反向解析，这里选择“no”。
- ☐ MaxStartups 5, 设置同时允许几个尚未登入的联机，当用户连上 SSH 但是尚未输入密码的时候就是所谓的联机，在这个联机中，为了保护主机，需要设定最大值，预设最多 10 个联机画面，而已经建立联机的不计算在这 10 个当中，其实设置 5 个已经够用了，这个设置可以防止对服务器进行恶意连接。
- ☐ MaxAuthTries 3, 设置最大失败尝试登录次数为 3，合理设置此值，可以防止攻击者穷举登录服务器。
- ☐ AllowUsers <用户名>, 指定允许通过远程访问的用户，多个用户以空格分隔。
- ☐ AllowGroups <组名>, 指定允许通过远程访问的用户组，多个用户组以空格分隔，当很多用户都需要通过 SSH 登录系统时，可将这些用户加入到一个用户组中。
- ☐ DenyUsers <用户名>, 指定禁止通过远程访问的用户，多个用户以空格分隔。
- ☐ DenyGroups <组名>, 指定禁止通过远程访问的用户组，多个用户组以空格分隔。

1.2.2 合理使用 shell 历史命令记录功能

在 Linux 下可通过 history 命令查看用户所有的历史操作记录，同时 shell 命令操作记录默认保存在用户目录下的 .bash_history 文件中，通过这个文件可以查询 shell 命令的执行历史，有助于运维人员进行系统审计和问题排查，同时，在服务器遭受黑客攻击后，也可以通过这个命令或文件查询黑客登录服务器所执行的历史命令操作。但是有时候黑客在入侵服务器后为了毁灭痕迹，可能会删除 .bash_history 文件，这就需要合理保护或备份 .bash_history 文件。下面介绍下 history 日志文件的安全配置方法。

默认的 history 命令只能查看用户历史操作记录，并不能区分每个用户操作命令的时间，

这点对于排查问题十分不便，不过可以通过下面的方法（加入四行内容）让 history 命令自动记录所有 shell 命令的执行时间，编辑 /etc/bashrc 文件：

```
HISTFILESIZE=4000
HISTSIZ=4000
HISTTIMEFORMAT='%F %T '
export HISTTIMEFORMAT
```

其中，HISTFILESIZE 定义了 .bash_history 文件中保存命令的记录总数，默认值是 1000，这里设置为 4000；HISTSIZ 定义了 history 命令输出的记录总数；HISTTIMEFORMAT 定义时间显示格式，这里的格式与 date 命令后的 “+%F %T” 是一致的；HISTTIMEFORMAT 作为 history 的时间变量将值传递给 history 命令。

通过这样的设置后，执行 history 命令，就会显示每条历史命令的详细执行时间，例如：

```
[root@server ~]# history
247 2013-10-05 17:16:28 vi /etc/bashrc
248 2013-10-05 17:16:28 top
249 2013-10-05 17:04:18 vmstat
250 2013-10-05 17:04:24 ps -ef
251 2013-10-05 17:16:29 ls -al
252 2013-10-05 17:16:32 lsattr
253 2013-10-05 17:17:16 vi /etc/profile
254 2013-10-05 17:19:32 date +%F %T
255 2013-10-05 17:21:06 lsof
256 2013-10-05 17:21:21 history
```

为了确保服务器的安全，保留 shell 命令的执行历史是非常有用的一条技巧。虽然 shell 有历史功能，但是这个功能并非针对审计目的而设计，因此很容易被黑客篡改或丢失。下面再介绍一种方法，可以实现详细记录登录过系统的用户、IP 地址、shell 命令以及详细操作时间等，并将这些信息以文件的形式保存在一个安全的地方，以供系统审计和故障排查。

将下面这段代码添加到 /etc/profile 文件中，即可实现上述功能。

```
#history
USER_IP=`who -u am i 2>/dev/null| awk '{print $NF}'|sed -e 's/[()]/g'`
HISTDIR=/usr/share/.history
if [ -z $USER_IP ]
then
USER_IP=`hostname`
fi
if [ ! -d $HISTDIR ]
then
mkdir -p $HISTDIR
```

```

chmod 777 $HISTDIR
fi
if [ ! -d $HISTDIR/${LOGNAME} ]
then
mkdir -p $HISTDIR/${LOGNAME}
chmod 300 $HISTDIR/${LOGNAME}
fi
export HISTSIZE=4000
DT=`date +%Y%m%d_%H%M%S`
export HISTFILE="$HISTDIR/${LOGNAME}/${USER_IP}.history.$DT"
export HISTTIMEFORMAT="%Y.%m.%d %H:%M:%S"
chmod 600 $HISTDIR/${LOGNAME}/*.history* 2>/dev/null

```

这段代码将每个用户的 shell 命令执行历史以文件的形式保存在 /usr/share/.history 目录中，每个用户一个文件夹，并且文件夹下的每个文件以 IP 地址加 shell 命令操作时间的格式命名。下面是 user01 用户执行 shell 命令的历史记录文件，基本效果如下：

```

[root@server user01]# pwd
/usr/share/.history/user01
[root@server user01]# ls -al
-rw----- 1 user01 wheel 56 Jul 6 17:07 192.168.12.12.history.20130706_164512
-rw----- 1 user01 wheel 43 Jul 6 17:42 192.168.12.12.history.20130706_172800
-rw----- 1 user01 wheel 22 Jul 7 12:05 192.168.12.19.history.20130707_111123
-rw----- 1 user01 wheel 22 Jul 8 13:41 192.168.12.20.history.20130708_120053
-rw----- 1 user01 wheel 22 Jul 1 15:28 192.168.12.186.history.20130701_150941
-rw----- 1 user01 wheel 22 Jul 2 19:47 192.168.12.163.history.20130702_193645
-rw----- 1 user01 wheel 22 Jul 3 12:38 192.168.12.19.history.20130703_120948
-rw----- 1 user01 wheel 22 Jul 3 19:14 192.168.12.134.history.20130703_183150

```

保存历史命令的文件夹目录要尽量隐蔽，避免被黑客发现后删除。

1.2.3 启用 tcp_wrappers 防火墙

tcp_wrappers 是一个用来分析 TCP/IP 封包的软件，类似的 IP 封包软件还有 iptables。Linux 默认安装了 tcp_wrappers。作为一个安全的系统，Linux 本身有两层安全防火墙，通过 IP 过滤机制的 iptables 实现第一层防护。iptables 防火墙通过直观地监视系统的运行状况，阻挡网络中的一些恶意攻击，保护整个系统正常运行，免遭攻击和破坏。如果通过了第一层防护，那么下一层防护就是 tcp_wrappers 了。通过 tcp_wrappers 可以实现对系统中提供的某些服务的开放与关闭、允许与禁止，从而更有效地保证系统安全运行。

tcp_wrappers 的使用很简单，仅仅有两个配置文件：/etc/hosts.allow 和 /etc/hosts.deny。

(1) 查看系统是否安装了 tcp_wrappers

```

[root@localhost ~]# rpm -q tcp_wrappers

```

```
tcp_wrappers-7.6-57.el6.x86_64
```

或者

```
[root@localhost ~]# rpm -qa | grep tcp
tcp_wrappers-7.6-57.el6.x86_64
tcp_wrappers-libs-7.6-57.el6.x86_64
tcpdump-4.0.0-3.20090921gitdf3cb4.2.el6.x86_64
```

如果有上面类似的输出，表示系统已经安装了 tcp_wrappers 模块。如果没有显示，可能没有安装，可以从 Linux 系统安装盘找到对应的 RPM 包进行安装。

(2) tcp_wrappers 防火墙的局限性

Linux 系统中的某个服务是否可以使用 tcp_wrappers 防火墙，取决于该服务是否应用了 libwrapped 库文件，如果应用了就可以使用 tcp_wrappers 防火墙。系统中默认的一些服务，如：sshd、portmap、sendmail、xinetd、vsftpd、tcpd 等都可以使用 tcp_wrappers 防火墙。

(3) tcp_wrappers 设定的规则

tcp_wrappers 防火墙的实现是通过 /etc/hosts.allow 和 /etc/hosts.deny 两个文件来完成的，首先看一下设定的格式：

```
service:host(s) [:action]
```

主要参数含义如下：

- ❑ service：代表服务名，例如 sshd、vsftpd、sendmail 等。
- ❑ host(s)：主机名或者 IP 地址，可以有多个，例如 192.168.12.0、www.ixdba.net。
- ❑ action：动作，符合条件后所采取的动作。

配置文件中常用的关键字有：

- ❑ ALL：所有服务或者所有 IP。
- ❑ ALL EXCEPT：从所有的服务或者所有 IP 中除去指定的。

例如：

```
ALL:ALL EXCEPT 192.168.12.189
```

表示除了 192.168.12.189 这台机器，任何机器执行所有服务时或被允许或被拒绝。

了解了设定语法后，下面就可以对服务进行访问限定。

例如，互联网上一台 Linux 服务器，实现的目标是：仅仅允许 222.61.58.88、61.186.232.58 以及域名 www.ixdba.net 通过 SSH 服务远程登录系统，下面介绍具体的设置过程。

首先设定允许登录的计算机，即配置 /etc/hosts.allow 文件，设置很简单，只要修改 /etc/hosts.allow（如果没有此文件，请自行建立）这个文件，即只需将下面规则加入 /etc/

hosts.allow 即可。

```
sshd: 222.61.58.88
sshd: 61.186.232.58
sshd: www.ixdba.net
```

接着设置不允许登录的机器，也就是配置 /etc/hosts.deny 文件。

一般情况下，Linux 会首先判断 /etc/hosts.allow 这个文件，如果远程登录的计算机满足文件 /etc/hosts.allow 设定，就不会再使用 /etc/hosts.deny 文件；相反，如果不满足 hosts.allow 文件设定的规则，就会使用 hosts.deny 文件，如果满足 hosts.deny 的规则，此主机就被限制为不可访问 Linux 服务器，如果也不满足 hosts.deny 的设定，此主机默认是可以访问 Linux 服务器的。因此，在设定好 /etc/hosts.allow 文件访问规则之后，只需设置 /etc/hosts.deny 为“所有计算机都不能登录状态”：

```
sshd:ALL
```

这样，一个简单的 tcp_wrappers 防火墙就设置完毕了。

1.3 文件系统安全

1.3.1 锁定系统重要文件

系统运维人员有时候可能会遇到通过 root 用户都不能修改或者删除某个文件的情况，产生这种情况的大部分原因可能是这个文件被锁定了。在 Linux 下锁定文件的命令是 chattr，通过这个命令可以修改 ext2、ext3、ext4 文件系统下文件属性，但是这个命令必须由超级用户 root 来执行。和这个命令对应的命令是 lsattr，后者用来查询文件属性。

通过 chattr 命令修改文件或者目录的文件属性能够提高系统的安全性，下面简单介绍下 chattr 和 lsattr 两个命令的用法。

chattr 命令的语法格式如下：

```
chattr [-RV] [-v version] [mode] 文件或目录
```

主要参数含义如下：

❑ -R：递归修改所有的文件及子目录。

❑ -V：详细显示修改内容，并打印输出。

其中 mode 部分用来控制文件的属性，常用参数如表 1-2 所示。

表 1-2 chattr 命令的常用参数

参数	含义
+	在原有参数设定基础上，追加参数
-	在原有参数设定基础上，移除参数
=	更新为指定参数
a	即 append，设定该参数后，只能向文件中添加数据，而不能删除。常用于服务器日志文件安全，只有 root 用户才能设置这个属性
c	即 compress，设定文件是否经压缩后再存储。读取时需要经过自动解压操作
i	即 immutable，设定文件不能被修改、删除、重命名、设定链接等，同时不能写入或新增内容。这个参数对于文件系统的安全设置有很大帮助
s	安全地删除文件或目录，即文件被删除后全部收回硬盘空间
u	与 s 参数相反，当设定为 u 时，系统会保留其数据块以便以后能够恢复删除这个文件

这些参数中，最常用到的是 a 和 i，a 参数常用于服务器日志文件安全设定，而 i 参数更为严格，不允许对文件进行任何操作，即使是 root 用户。

lsattr 用来查询文件属性，用法比较简单，其语法格式如下：

```
lsattr [-adlrsv] 文件或目录
```

常用参数如表 1-3 所示。

表 1-3 lsattr 命令的常用参数

参数	含义
-a	列出目录中的所有文件，包括以“.”开头的文件
-d	显示指定目录的属性
-R	以递归的方式列出目录下所有文件及子目录以及属性值
-v	显示文件或目录版本

在 Linux 系统中，如果一个用户以 root 的权限登录或者某个进程以 root 的权限运行，那么它的使用权限就不再有任何的限制了。因此，攻击者通过远程或者本地攻击手段获得了系统的 root 权限将是一场灾难。在这种情况下，文件系统将是保护系统安全的最后一道防线，合理的属性设置可以最大限度地降低攻击者对系统的破坏程度，通过 chattr 命令锁定系统一些重要的文件或目录，是保护文件系统安全最直接、最有效的手段。

对一些重要的目录和文件可以加上“i”属性，常见的文件和目录有：

```
chattr -R +i /bin /boot /lib /sbin
chattr -R +i /usr/bin /usr/include /usr/lib /usr/sbin
chattr +i /etc/passwd
chattr +i /etc/shadow
chattr +i /etc/hosts
chattr +i /etc/resolv.conf
chattr +i /etc/fstab
```

```
chattr +i /etc/sudoers
```

对一些重要的日志文件可以加上“a”属性，常见的有：

```
chattr +a /var/log/messages
```

```
chattr +a /var/log/wtmp
```

对重要的文件进行加锁，虽然能够提高服务器的安全性，但是也会带来一些不便，例如，在软件的安装、升级时可能需要去掉有关目录和文件的 immutable 属性和 append-only 属性，同时，对日志文件设置了 append-only 属性，可能会使日志轮换 (logrotate) 无法进行。因此，在使用 chattr 命令前，需要结合服务器的应用环境来权衡是否需要设置 immutable 属性和 append-only 属性。

另外，虽然通过 chattr 命令修改文件属性能够提高文件系统的安全性，但是它并不适合所有的目录。chattr 命令不能保护 /、/dev、/tmp、/var 等目录。

根目录不能有不可修改属性，因为如果根目录具有不可修改属性，那么系统根本无法工作：/dev 在启动时，syslog 需要删除并重新建立 /dev/log 套接字设备，如果设置了不可修改属性，那么可能出问题；/tmp 目录会有很多应用程序和系统程序需要在这个目录下建立临时文件，也不能设置不可修改属性；/var 是系统和程序的日志目录，如果设置为不可修改属性，那么系统将无法写日志，所以也不能通过 chattr 命令保护。

虽然保护 /dev、/tmp 等目录的安全性无法通过 chattr 命令实现，但是可以通过另外的方法实现，这些将在下一小节详细介绍。

1.3.2 文件权限检查和修改

不正确的权限设置直接威胁系统的安全，因此运维人员应该能及时发现这些不正确的权限设置，并立刻修正，防患于未然。下面列举几种查找系统不安全权限的方法。

(1) 查找系统中任何用户都有写权限的文件或目录

查找文件：`find / -type f -perm -2 -o -perm -20 |xargs ls -al`

查找目录：`find / -type d -perm -2 -o -perm -20 |xargs ls -ld`

(2) 查找系统中所有含“s”位的程序

```
find / -type f -perm -4000 -o -perm -2000 -print | xargs ls -al
```

含有“s”位权限的程序对系统安全威胁很大，通过查找系统中所有具有“s”位权限的程序，可以把某些不必要的“s”位权限的程序去掉，这样可以防止用户滥用权限或提升权限的可能性。

(3) 检查系统中所有 suid 及 sgid 文件

```
find / -user root -perm -2000 -print -exec md5sum {} \;
```

```
find / -user root -perm -4000 -print -exec md5sum {} \;
```

将检查的结果保存到文件中,可在以后的系统检查中作为参考。

(4) 检查系统中没有属主的文件

```
find / -nouser -o -nogroup
```

没有属主的孤儿文件比较危险,往往成为黑客利用的工具,因此在找到这些文件后,要么删除,要么修改文件的属主,使其处于安全状态。

1.3.3 /tmp、/var/tmp、/dev/shm 安全设定

在 Linux 系统中,用来存放临时文件主要有两个目录或分区,分别是 /tmp 和 /var/tmp。存储临时文件的目录或分区有个共同点,就是所有用户可读写、可执行,这就为系统留下了安全隐患。攻击者可以将病毒或者木马脚本放到临时文件的目录下进行信息收集或伪装,严重影响服务器的安全,此时,如果修改临时目录的读写执行权限,还有可能影响系统上应用程序正常运行,因此,如果要兼顾两者,就需要对这两个目录或分区进行特殊设置。

/dev/shm 是 Linux 下的一个共享内存设备,在 Linux 启动的时候系统默认会加载 /dev/shm,被加载的 /dev/shm 使用的是 tmpfs 文件系统,而 tmpfs 是一个内存文件系统,存储到 tmpfs 文件系统的数据会完全驻留在 RAM 中,这样通过 /dev/shm 就可以直接操控系统内存,这将非常危险,因此如何保证 /dev/shm 安全也至关重要。

对于 /tmp 的安全设置,需要看 /tmp 是一个独立磁盘分区,还是一个根分区下的文件夹。如果 /tmp 是一个独立的磁盘分区,那么设置非常简单,修改 /etc/fstab 文件中 /tmp 分区对应的挂载属性,加上 nosuid、noexec、nodev 三个选项即可,修改后的 /tmp 分区挂载属性类似如下:

```
LABEL=/tmp      /tmp      ext3      rw,nosuid,noexec,nodev    0 0
```

其中,nosuid、noexec、nodev 选项表示不允许任何 suid 程序,在这个分区不能执行任何脚本等程序,以及不存在设备文件。

在挂载属性设置完成后,重新挂载 /tmp 分区,保证设置生效。

对于 /var/tmp,如果是独立分区,安装 /tmp 的设置方法是修改 /etc/fstab 文件即可;如果是 /var 分区下的一个目录,那么可以将 /var/tmp 目录下所有数据移动到 /tmp 分区下,然后在 /var 下做一个指向 /tmp 的软连接即可。也就是执行如下操作:

```
[root@server ~]# mv /var/tmp/* /tmp
```

```
[root@server ~]# ln -s /tmp /var/tmp
```

如果 /tmp 是根目录下的一个目录，那么设置稍微复杂，可以通过创建一个 loopback 文件系统来利用 Linux 内核的 loopback 特性将文件系统挂载到 /tmp 下，然后在挂载时指定限制加载选项即可。一个简单的操作示例如下：

```
[root@server ~]# dd if=/dev/zero of=/dev/tmpfs bs=1M count=10000
[root@server ~]# mke2fs -j /dev/tmpfs
[root@server ~]# cp -av /tmp /tmp.old
[root@server ~]# mount -o loop,noexec,nosuid,rw /dev/tmpfs /tmp
[root@server ~]# chmod 1777 /tmp
[root@server ~]# mv -f /tmp.old/* /tmp/
[root@server ~]# rm -rf /tmp.old
```

最后，编辑 /etc/fstab，添加如下内容，以便系统在启动时自动加载 loopback 文件系统：

```
/dev/tmpfs /tmp ext3 loop,nosuid,noexec,rw 0 0
```

为了验证挂载时指定限制加载选项是否生效，可以在 /tmp 分区创建一个 shell 文件，操作如下：

```
[root@tc193 tmp]# ls -al|grep shell
-rwxr-xr-x  1 root root    22 Oct  6 14:58 shell-test.sh
[root@server ~]# pwd
/tmp
[root@tc193 tmp]# ./shell-test.sh
-bash: ./shell-test.sh: Permission denied
```

可以看出，虽然文件有可执行属性，但是在 /tmp 分区已经无法执行任何文件了。

最后介绍如何修改 /dev/shm 的安全设置。由于 /dev/shm 是一个共享内存设备，因此也可以通过修改 /etc/fstab 文件设置而实现，在默认情况下，/dev/shm 通过 defaults 选项来加载，这样保证其安全性是不够的，需要修改 /dev/shm 的挂载属性，操作如下：

```
tmpfs /dev/shm tmpfs defaults,nosuid,noexec,rw 0 0
```

通过这种方式限制了任何 suid 程序，同时也限制了 /dev/shm 的可执行权限，系统安全性得到进一步提升。

1.4 系统软件安全管理

根据安全机构权威统计，80% 以上的服务器遭受攻击都是由于服务器上的系统软件或者应用程序的漏洞所导致，黑客通过这些软件的漏洞，可以很轻易地攻入服务器，由此可见，软件的漏洞已经成为安全的中中之重。作为一名运维人员，虽然无法保证所有应用程序的安全，但是对于系统软件的安全，要有定期检查并试图修复漏洞的意识。修复漏洞最

常见的办法就是升级软件，将软件始终保持在最新状态，可以在一定程度上保证系统安全。

在 Linux 下软件的升级可以分为自动升级和手动升级两种方式。自动升级一般是在有授权的 Linux 发行版或者免费 Linux 发行版下进行的，只要输入升级命令，系统会自动完成升级工作，无需人工干预。

手动升级是指有针对性地进行某个系统软件的升级，例如升级系统的 SSH 登录工具、gcc 编译工具等。手动升级其实就是通过 RPM 包实现软件更新的，通过这种方式在升级软件时可能会遇到软件之间的依赖关系，升级相对比较麻烦。

1.4.1 软件自动升级工具 yum

yum 是 yellow dog updaters modified 的缩写，yellow dog（黄狗）也是 Linux 的一个发行版本，只不过 Redhat 公司将这种升级技术利用到自己的发行版上就形成了现在的 yum。yum 是进行 Linux 自动升级常用的一个工具，通过 yum 工具配合互联网即可实现自动升级系统。例如，一个经过授权的 Redhat Linux 操作系统，或者一个 CentOS Linux 系统，只要该系统能连接互联网，输入 yum update 即可实现系统自动升级。通过 yum 进行系统升级实质是使用 yum 命令下载指定远程互联网主机上的 RPM 软件包，然后自动进行安装，同时解决各个软件之间的依赖关系。

1.4.2 yum 的安装与配置

1. yum 的安装

检查是否已经安装 yum：

```
[root@localhost ~]# rpm -qa|grep yum
```

如果没有任何显示，表示系统中还没有安装 yum 工具。yum 安装包在 CentOS 系统光盘中可以找到，执行如下命令进行安装：

```
[root@localhost ~]# rpm -ivh yum-*.noarch.rpm
```

安装 yum 需要 python-elementtree、python-sqlite、urlgrabber、yumconf 等软件包的支持，这些软件包在 CentOS Linux 系统安装光盘中均可找到。如果在安装 yum 过程中出现软件包之间的依赖性，只需按照依赖提示寻找相应软件包安装即可，直到 yum 包安装成功。

2. yum 的配置

安装完 yum 工具安装后，接下来的工作是配置 yum。yum 的配置文件有主配置文件 /etc/yum.conf 和资源库配置目录 /etc/yum.repos.d。安装 yum 后，默认的一些资源库配置可

能无法使用，因此需要进行修改。下面介绍 `/etc/yum.repos.d/CentOS-Base.repo` 资源库配置文件的内容以及各项的详细含义。

```
[root@localhost ~]#more /etc/yum.repos.d/CentOS-Base.repo
[base]
name=CentOS-$releasever - Base

mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&repo=os

gpgcheck=1

[updates]
# 下面这段是 updates 更新模块要用到的部分配置
name=CentOS-$releasever - Updates
mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&repo=
updates gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-6

# 下面这段指定的是有用的额外软件包部分 (extras) 配置
[extras]
name=CentOS-$releasever - Extras
mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&repo=
extras gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-6

# 下面这段指定的是扩展的额外软件包部分 (centosplus) 配置
[centosplus]
name=CentOS-$releasever - Plus
mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&repo=
centosplus gpgcheck=1
enabled=0
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-6

# 下面这段是 contrib 配置部分
[contrib]
name=CentOS-$releasever - Contrib
mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&repo=
contrib gpgcheck=1
enabled=0
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-6
```

在上面这个配置中，几个常用关键字的含义介绍如下：

- ❑ `name` 表示发行版的名称，其格式表示“操作系统名和释出版本”，“Base”表明此段寻找的是 Base 包信息。
- ❑ `mirrorlist` 表示 yum 在互联网上查找升级文件的 URL 地址。其中“\$basearch”代

表系统的硬件架构，如“i386”、“x86-64”等，同时，yum 在资源更新时，会检查 baseurl/repodata/repomd.xml 文件。“repomd.xml”是一个索引文件，它的作用是提供更新 RPM 包文件的下载信息和 SHA 校验值。“repomd.xml”包括了三个文件，分别为 other.xml.gz、filelists.xml.gz 和 primary.xml.gz，表示的含义依次是其他更新包列表、更新文件集中列表和主要更新包列表。

❑ gpgcheck 表示是否启用 gpg 检查，1 表示启用，0 表示不启用校验。如果启用，就需要在配置文件中注明 GPG-RPM-KEY 的位置。

❑ gpgkey 用来指定 GPG 密钥的地址。

1.4.3 yum 的特点与基本用法

1. yum 的特点

yum 的特点如下：

❑ 安装方便，自动解决增加或删除 RPM 包时遇到的依赖性问题。

❑ 可以同时配置多个资源库 (Repository)。

❑ 配置文件简单明了 (/etc/yum.conf、/etc/yum.repos.d/CentOS-Base.repo)。

❑ 保持与 RPM 数据库的一致性。



注意 yum 会自动下载所有所需的升级资源包并默认放置在 /var/cache/yum 目录下，当第一次使用 yum 或 yum 资源库更新时，软件升级所需的时间可能较长。

2. yum 的基本用法

yum 的基本用法主要有 4 种，下面分别介绍。

(1) 通过 yum 安装和删除 RPM 包

安装 RPM 包，如 dhcp：

```
[root@localhost ~]# yum install dhcp
```

删除 RPM 包，包括与该包有依赖性的包：

```
[root@localhost ~]# yum remove licq
```

注意，同时会提示删除 licq-gnome、licq-qt、licq-text。

(2) 通过 yum 工具更新软件包

检查可更新的 RPM 包：

```
[root@localhost ~]# yum check-update
```

更新所有的 RPM 包:

```
[root@localhost ~]# yum update
```

更新指定的 RPM 包, 如更新 kernel 和 kernel source:

```
[root@localhost ~]# yum update kernel kernel-source
```

大规模的版本升级, 与 yum update 不同的是, 陈旧的包也会升级:

```
[root@localhost ~]# yum upgrade
```

(3) 通过 yum 查询 RPM 包信息

列出资源库中所有可以安装或更新的 RPM 包的信息:

```
[root@localhost ~]# yum info
```

列出资源库中特定的可以安装或更新以及已经安装的 RPM 包的信息:

```
[root@localhost ~]# yum info vsftpd
```

```
[root@localhost ~]# yum info perl*
```

注意, 可以在 RPM 包名中使用匹配符, 如上面的例子列出所有以 perl 开头的 RPM 包的信息。

列出资源库中所有可以更新的 RPM 包的信息:

```
[root@localhost ~]# yum info updates
```

列出已经安装的所有的 RPM 包的信息:

```
[root@localhost ~]# yum info installed
```

列出已经安装但是不包含在资源库中的 RPM 包的信息:

```
[root@localhost ~]# yum info extras
```

注意, 也就是通过其他网站下载安装的 RPM 包的信息。

列出资源库中所有可以更新的 RPM 包:

```
[root@localhost ~]# yum list updates
```

列出已经安装的所有 RPM 包:

```
[root@localhost ~]# yum list installed
```

列出已经安装的但不包含在资源库中的 RPM 包:

```
[root@localhost ~]# yum list extras
```


注意，也就是通过其他网站下载安装的 RPM 包。

列出资源库中所有可以安装或更新的 RPM 包：

```
[root@localhost ~]# yum list
```

列出资源库中特定的可以安装或更新以及已经安装的 RPM 包：

```
[root@localhost ~]# yum list sendmail
```

```
[root@localhost ~]# yum list gcc*
```

注意，可以在 RPM 包名中使用匹配符，如上面的例子列出了所有以 gcc 开头的 RPM 包。搜索匹配特定字符的 RPM 包的详细信息：

```
[root@localhost ~]# yum search wget
```

注意，可以通过“search”在 RPM 包名、包描述中搜索。

搜索包含特定文件名的 RPM 包：

```
[root@localhost ~]# yum provides realplay
```

(4) 通过 yum 操作暂存信息 (/var/cache/yum)

清除暂存的 RPM 包文件：

```
[root@localhost ~]# yum clean packages
```

清除暂存的 RPM 头文件：

```
[root@localhost ~]# yum clean headers
```

清除暂存中旧的 RPM 头文件：

```
[root@localhost ~]# yum clean oldheaders
```

清除暂存中旧的 RPM 头文件和包文件：

```
[root@localhost ~]# yum clean
```

或

```
[root@localhost ~]# yum clean all
```

注意，上面的两个命令相当于 yum clean packages + yum clean oldheaders。

1.4.4 几个不错的 yum 源

由于 CentOS 系统自带的官方 yum 源中去除了很多有版权争议的软件，所以可使用的软件种类并不丰富，而且软件版本普遍较低，对于软件 bug 修复更新也很慢，有时候需要

使用最新稳定版本的软件时，可能需要手动进行软件更新，操作比较麻烦。下面介绍几个不错的 yum 源，以供软件升级和漏洞修复使用。

(1) EPEL

EPEL 全称是企业版 Linux 附加软件包，是由特别兴趣小组创建、维护并管理，针对红帽企业版 Linux(RHEL) 及其衍生发行版（例如 CentOS、Scientific Linux）的一个高质量附加软件包项目。其官方网址为：<http://fedoraproject.org/wiki/EPEL/zh-cn>。EPEL 的软件包不会与企业版 Linux 官方源中的软件包发生冲突，或者互相替换文件，因此可以放心使用。

EPEL 包含一个名为“epel-release”的包，这个包包含了 EPEL 源的 gpg 密钥和软件源信息。可以通过 yum 命令将这个软件包安装到企业级 Linux 发行版上，这样就可以使用最全面、最稳定的 Linux 软件包了。除了 epel-release 源，还有一个名为“epel-testing”的源，这个源包含最新的测试软件包，其版本很新但是安装有风险，可以根据情况自行斟酌使用。

相关的 EPEL 软件包可以从 EPEL 官方网站下载，现在有针对性企业版 5 和企业版 6 的两个 RPM 包，读者可根据系统环境进行下载使用。

(2) RPMForge

RPMForge 是一个第三方的软件源仓库，也是 CentOS 官方社区推荐的第三方 yum 源，它为 CentOS 系统提供了超过 10 000 个软件包，被 CentOS 社区认定为最安全也最稳定的一个软件仓库。但是由于这个安装源不是 CentOS 本身的组成部分，因此要使用 RPMForge，需要手动下载并安装。RPMForge 的官方网站是 <http://repoforge.org/>，可以在 <http://pkgs.repoforge.org/rpmforge-release/> 下载 RHEL/Centos 各个版本的“rpmforge-release”包，这样就可以使用 RPMForge 中的丰富软件了。

1.5 Linux 后门入侵检测工具

rootkit 是 Linux 平台下最常见的一种木马后门工具，它主要通过替换系统文件来达到入侵和隐蔽的目的，这种木马比普通木马后门更加危险和隐蔽，普通的检测工具和检查手段很难发现这种木马。rootkit 攻击能力极强，对系统的危害很大，它通过一套工具来建立后门和隐藏行迹，从而让攻击者保住权限，以使它在任何时候都可以使用 root 权限登录系统。

rootkit 主要有两种类型：文件级别和内核级别，下面分别进行简单介绍。

1. 文件级别 rootkit

文件级别的 rootkit 一般是通过程序漏洞或者系统漏洞进入系统后，通过修改系统的重要文件来达到隐藏自己的目的。在系统遭受 rootkit 攻击后，合法的文件被木马程序替代，

变成了外壳程序，而其内部是隐藏着的后门程序。通常容易被 rootkit 替换的系统程序有 login、ls、ps、ifconfig、du、find、netstat 等，其中 login 程序是最经常被替换的，因为当访问 Linux 时，无论是通过本地登录还是远程登录，/bin/login 程序都会运行，系统将通过 /bin/login 来收集并核对用户的账号和密码，而 rootkit 就是利用这个程序的特点，使用一个带有根权限后门密码的 /bin/login 来替换系统的 /bin/login，这样攻击者通过输入设定好的密码就能轻松进入系统。此时，即使系统管理员修改 root 密码或者清除 root 密码，攻击者还是一样能通过 root 用户登录系统。通常攻击者在进入 Linux 系统后，会进行一系列的攻击动作，最常见的是安装嗅探器收集本机或者网络中其他服务器的重要数据。在默认情况下，Linux 中也有一些系统文件会监控这些工具动作，例如 ifconfig 命令，所以，攻击者为了避免被发现，会想方设法替换其他系统文件，常见的就是 ls、ps、ifconfig、du、find、netstat 等。如果这些文件都被替换，那么在系统层面就很难发现 rootkit 已经在系统中运行了。

这就是文件级别的 rootkit，对系统维护威胁很大，目前最有效的防御方法是定期对系统重要文件的完整性进行检查，如果发现文件被修改或者替换，那么很可能系统已经遭受了 rootkit 入侵。检查文件完整性的工具很多，常见的有 Tripwire、aide 等，可以通过这些工具定期检查文件系统的完整性，以检测系统是否被 rootkit 入侵。

2. 内核级别的 rootkit

内核级别 rootkit 是比文件级别 rootkit 更高级的一种入侵方式，它可以使攻击者获得对系统底层的完全控制权，此时攻击者可以修改系统内核，进而截获运行程序向内核提交的命令，并将其重定向到入侵者所选择的程序并运行此程序，也就是说，当用户要运行程序 A 时，被入侵者修改过的内核会假装执行 A 程序，而实际上却执行了程序 B。

内核级别 rootkit 主要依附在内核上，它并不对系统文件做任何修改，因此一般的检测工具很难检测到它的存在，这样一旦系统内核被植入 rootkit，攻击者就可以对系统为所欲为而不被发现。目前针对内核级的 rootkit 还没有很好的防御工具，因此，做好系统安全防范就非常重要，将系统维持在最小权限内工作，只要攻击者不能获取 root 权限，就无法在内核中植入 rootkit。

1.5.1 rootkit 后门检测工具 chkrootkit

chkrootkit 是 Linux 系统下查找并检测 rootkit 后门的一个工具，它的官方网站是：<http://www.chkrootkit.org/>。chkrootkit 没有包含在官方的 CentOS 源中，因此要采取手动编译的方法来安装，不过这种安装方法也更加安全。下面简单介绍下 chkrootkit 的安装过程。

1. 准备 gcc 编译环境

对于 CentOS 系统，需要安装 gcc 编译环境，执行下述三个命令：

```
[root@server ~]# yum -y install gcc
[root@server ~]# yum -y install gcc-c++
[root@server ~]# yum -y install make
```

2. 安装 chkrootkit

为了安全起见，建议直接从官方网站下载 chkrootkit 源码，然后进行安装，操作如下：

```
[root@server ~]# tar zxvf chkrootkit.tar.gz
[root@server ~]# cd chkrootkit-*
[root@server chkrootkit-0.50]# make sense
# 注意，上面的编译命令为 make sense
[root@server chkrootkit-0.50]# cd ..
[root@server ~]# cp -r chkrootkit-* /usr/local/chkrootkit
[root@server ~]# rm -rf chkrootkit-*
```

3. 使用 chkrootkit

安装完的 chkrootkit 程序位于 /usr/local/chkrootkit 目录下，执行如下命令即可显示 chkrootkit 的详细用法：

```
[root@server chkrootkit]# /usr/local/chkrootkit/chkrootkit -h
```

chkrootkit 各个参数的含义如表 1-4 所示。

表 1-4 chkrootkit 各个参数的含义

参数	含义
-h	显示帮助信息
-v	显示版本信息
-l	显示测试内容
-d	debug 模式，显示检测过程的相关命令程序
-q	安静模式，只显示有问题的内容
-x	高级模式，显示所有检测结果
-r dir	设置指定的目录为根目录
-p dir1:dir2:dirN	指定 chkrootkit 检测时使用系统命令的目录
-n	跳过 NFS 连接的目录

chkrootkit 的使用比较简单，直接执行 chkrootkit 命令即可自动开始检测系统。下面是某个系统的检测结果：

```
[root@server chkrootkit]# /usr/local/chkrootkit/chkrootkit
Checking `ifconfig'... INFECTED
```



```

Checking `ls'... INFECTED
Checking `login'... INFECTED
Checking `netstat'... INFECTED
Checking `ps'... INFECTED
Checking `top'... INFECTED
Checking `sshd'... not infected
Checking `syslogd'... not tested
Checking `tar'... not infected
Checking `tcpd'... not infected
Checking `tcpdump'... not infected
Checking `telnetd'... not found

```

从输出结果可以看出，此系统的 ifconfig、ls、login、netstat、ps 和 top 命令已经被感染。针对被感染 rootkit 的系统，最安全而有效的方法就是备份数据并重新安装系统。

4. chkrootkit 的缺点

chkrootkit 在检查 rootkit 的过程中使用了部分系统命令，因此，如果服务器被黑客入侵，那么依赖的系统命令可能也已经被入侵者替换，此时 chkrootkit 的检测结果将变得完全不可信。为了避免 chkrootkit 的这个问题，可以在服务器对外开放前，事先将 chkrootkit 使用的系统命令进行备份，在需要的时候使用备份的原始系统命令让 chkrootkit 对 rootkit 进行检测。这个过程可以通过下面的操作实现：

```

[root@server ~]# mkdir /usr/share/.commands
[root@server ~]# cp `which --skip-alias awk cut echo find egrep id head ls netstat ps
strings sed uname` /usr/share/.commands
[root@server ~]# /usr/local/chkrootkit/chkrootkit -p /usr/share/.commands/
[root@server share]# cd /usr/share/
[root@server share]# tar zcvf commands.tar.gz .commands
[root@server share]# rm -rf commands.tar.gz

```

上面这段操作是在 /usr/share/ 下建立了一个 .commands 隐藏文件，然后将 chkrootkit 使用的系统命令进行备份到这个目录下。为了安全起见，可以将 .commands 目录压缩打包，然后下载到一个安全的地方进行备份，以后如果服务器遭受入侵，就可以将这个备份上传到服务器任意路径下，然后通过 chkrootkit 命令的“-p”参数指定这个路径进行检测即可。

1.5.2 rootkit 后门检测工具 RKHunter

RKHunter 是专业检测系统是否感染 rootkit 的一个工具，它通过执行一系列的脚本来确认服务器是否已经感染 rootkit。在官方资料中，RKHunter 可以做的事情有：

- ❑ MD5 校验测试，检测文件是否有改动
- ❑ 检测 rootkit 使用的二进制和系统工具文件

- ❑ 检测特洛伊木马程序的特征码
- ❑ 检测常用程序的文件属性是否异常
- ❑ 检测系统相关的测试
- ❑ 检测隐藏文件
- ❑ 检测可疑的核心模块 LKM
- ❑ 检测系统已启动的监听端口

下面详细讲述下 RKHunter 的安装与使用。

1. 安装 RKHunter

RKHunter 的官方网站为：http://www.rootkit.nl/projects/rootkit_hunter.html，建议从这个网站下载 RKHunter，本书下载使用的版本是 rkhunter-1.4.0.tar.gz。RKHunter 的安装非常简单，具体过程如下：

```
[root@server ~]# ls
rkhunter-1.4.0.tar.gz
[root@server ~]# pwd
/root
[root@server ~]# tar -zxvf rkhunter-1.4.0.tar.gz
[root@server ~]# cd rkhunter-1.4.0
[root@server rkhunter-1.4.0]# ./installer.sh --layout default --install
```

这里采用 RKHunter 的默认安装方式，rkhunter 命令被安装到 /usr/local/bin 目录下。

2. 使用 rkhunter 命令

rkhunter 命令的参数较多，但是使用非常简单，直接运行 rkhunter 即可显示此命令的用法。下面简单介绍下 rkhunter 常用的几个参数选项。

```
[root@server ~]# /usr/local/bin/rkhunter-help
```

rkhunter 的常用参数以及含义如表 1-5 所示。

表 1-5 rkhunter 常用参数的含义

参数	含义
-c, --check	必选参数，表示检测当前系统
--configfile <file>	使用特定的配置文件
--cronjob	作为 cron 任务定期运行
--sk, --skip-keypress	自动完成所有检测，跳过键盘输入
--summary	显示检测结果的统计信息
--update	检测更新内容
-V, --version	显示版本信息
--versioncheck	检测最新版本

下面是通过 RKHunter 对某个系统的检测示例。

```
[root@server rkhunter-1.4.0]# /usr/local/bin/rkhunter -c
[ Rootkit Hunter version 1.4.0 ]
# 下面是第一部分，先进行系统命令的检查，主要是检测系统的二进制文件，因为这些文件最容易被 rootkit
# 攻击。显示 OK 字样表示正常，显示 Warning 表示有异常，需要引起注意，而显示“Not found”字样，一
# 般无需理会
Checking system commands...

Performing 'strings' command checks
  Checking 'strings' command [ OK ]

Performing 'shared libraries' checks
  Checking for preloading variables [ None found ]
  Checking for preloaded libraries [ None found ]
  Checking LD_LIBRARY_PATH variable [ Not found ]

Performing file properties checks
  Checking for prerequisites [ Warning ]
  /usr/local/bin/rkhunter [ OK ]
  /sbin/chkconfig [ OK ]
....(略)....
[Press <ENTER> to continue]
# 下面是第二部分，主要检测常见的 rootkit 程序，显示“Not found”表示系统未感染此 rootkit
Checking for rootkits...
Performing check of known rootkit files and directories
  55808 Trojan - Variant A [ Not found ]
  ADM Worm [ Not found ]
  AjaKit Rootkit [ Not found ]
  Adore Rootkit [ Not found ]
aPa Kit [ Not found ]
  Apache Worm [ Not found ]
  Ambient (ark) Rootkit [ Not found ]
  Balaar Rootkit [ Not found ]
  BeastKit Rootkit [ Not found ]
beX2 Rootkit [ Not found ]
  BOBKit Rootkit [ Not found ]
....(略)....
[Press <ENTER> to continue]
# 下面是第三部分，主要是一些特殊或附加的检测，例如对 rootkit 文件或目录检测、对恶意软件检测以及
# 对指定的内核模块检测
Performing additional rootkit checks
  Suckit Rootkit additional checks [ OK ]
  Checking for possible rootkit files and directories [ None found ]
  Checking for possible rootkit strings [ None found ]

Performing malware checks
```

```

[ OK ]
Checking running processes for suspicious files [ None found ]
Checking for login backdoors [ None found ]
Checking for suspicious directories [ None found ]
Checking for sniffer log files [ None found ]
Performing Linux specific checks
Checking loaded kernel modules [ OK ]
Checking kernel module names [ OK ]
[Press <ENTER> to continue]
# 下面是第四部分，主要对网络、系统端口、系统启动文件、系统用户和组配置、SSH 配置、文件系统等进
# 检测
Checking the network...
Performing checks on the network ports
Checking for backdoor ports [ None found ]
Performing checks on the network interfaces
Checking for promiscuous interfaces [ None found ]
Checking the local host...
Performing system boot checks
Checking for local host name [ Found ]
Checking for system startup files [ Found ]
Checking system startup files for malware [ None found ]
Performing group and account checks
Checking for passwd file [ Found ]
Checking for root equivalent (UID 0) accounts [ None found ]
Checking for passwordless accounts [ None found ]
....(略)....
[Press <ENTER> to continue]
# 下面是第五部分，主要是对应用程序版本进行检测
Checking application versions...
Checking version of GnuPG [ OK ]
Checking version of OpenSSL [ Warning ]
Checking version of OpenSSH [ OK ]
# 下面是第六部分，这一部分其实是上面输出的一个总结，通过这个总结，可以大概了解服务器目录的安全状态
System checks summary
=====
File properties checks...
Required commands check failed
Files checked: 137
Suspect files: 4
Rootkit checks...
Rootkits checked : 311
Possible rootkits: 0
Applications checks...

```



```
Applications checked: 3
```

```
Suspect applications: 1
```

```
The system checks took: 6 minutes and 41 seconds
```

在 Linux 终端使用 RKHunter 来检测，最大的好处在于每项的检测结果都使用不同的颜色显示，如果是绿色的表示没有问题，如果是红色的，那就要引起关注了。另外，在上面执行检测的过程中，在每个部分检测完成后，需要以 Enter 键来继续。如果要让程序自动运行，可以执行如下命令：

```
[root@server ~]# /usr/local/bin/rkhunter --check --skip-keypress
```

同时，如果想让检测程序每天定时运行，那么可以在 /etc/crontab 中加入如下内容：

```
09 3 * * * root /usr/local/bin/rkhunter --check --cronjob
```

这样，RKHunter 检测程序就会在每天的 3:09 运行一次。

1.6 服务器遭受攻击后的处理过程

安全总是相对的，再安全的服务器也有可能遭受到攻击。作为一名安全运维人员，要把握的原则是：尽量做好系统安全防护，修复所有已知的危险行为，同时，在系统遭受攻击后能够迅速有效地处理攻击行为，最大限度地降低攻击对系统产生的影响。

1.6.1 处理服务器遭受攻击的一般思路

系统遭受攻击并不可怕，可怕的是面对攻击束手无策，下面就详细介绍下在服务器遭受攻击后的一般处理思路。

1. 切断网络

所有的攻击都来自于网络，因此，在得知系统正遭受黑客的攻击后，首先要做的就是断开服务器的网络连接，这样除了能切断攻击源之外，也能保护服务器所在网络的其他主机。

2. 查找攻击源

可以通过分析系统日志或登录日志文件，查看可疑信息，同时也要查看系统都打开了哪些端口，运行哪些进程，并通过这些进程分析哪些是可疑的程序。这个过程要根据经验和综合判断能力进行追查和分析。后面的章节会详细介绍这个过程的处理思路。

3. 分析入侵原因和途径

既然系统遭到入侵，那么原因是多方面的，可能是系统漏洞，也可能是程序漏洞，一

定要查清楚是哪个原因导致的，并且还要查清楚遭到攻击的途径，找到攻击源，因为只有知道了遭受攻击的原因和途径，才能在删除攻击源的同时进行漏洞的修复。

4. 备份用户数据

在服务器遭受攻击后，需要立刻备份服务器上的用户数据，同时也要查看这些数据中是否隐藏着攻击源。如果攻击源在用户数据中，一定要彻底删除，然后将用户数据备份到一个安全的地方。

5. 重新安装系统

永远不要认为自己能彻底清除攻击源，因为没有人能比黑客更了解攻击程序，在服务器遭到攻击后，最安全也最简单的方法就是重新安装系统，因为大部分攻击程序都会依附在系统文件或者内核中，所以重新安装系统才能彻底清除攻击源。

6. 修复程序或系统漏洞

在发现系统漏洞或者应用程序漏洞后，首先要做的就是修复系统漏洞或者更改程序bug，只有将程序的漏洞修复完毕才能正式在服务器上运行。

7. 恢复数据和连接网络

将备份的数据重新复制到新安装的服务器上，然后开启服务，最后将服务器的网络连接开启，对外提供服务。

1.6.2 检查并锁定可疑用户

在发现服务器遭受攻击后，首先要切断网络连接，但是在有些情况下，比如无法马上切断网络连接时，就必须登录系统查看是否有可疑用户，如果有可疑用户登录了系统，那么需要马上将这个用户锁定，然后中断此用户的远程连接。

1. 登录系统查看可疑用户

通过 root 用户登录，然后执行“w”命令即可列出所有登录过系统的用户，如图 1-11 所示。

```
[root@server ~]# w
19:12:46 up 12 days, 8:31, 28 users, load average: 0.56, 0.67, 0.67
USER      TTY      FROM          LOGIN@      IDLE   JCPU   PCPU   WHAT
nobody    pts/3    122.21.161.189 Fri05       2days  0.04s  0.04s  -bash
user01    pts/4    189.22.1.90   26Sep13     2days  1.03s  0.00s  sshd: user01 [priv]
user02    pts/16   124.33.5.67   26Sep13     2days  16.61s 16.54s /usr/local/java/bin/java -Xmx1000m
user100   pts/29   192.201.12.189 28Sep13     2days  0.88s  0.01s  sshd: user100 [priv]
user03    pts/23   218.60.96.13  26Sep13     2days  39.47s 0.01s  sshd: user03 [priv]
```

图 1-11 查看所有登录过系统的用户

通过这个输出可以检查是否有可疑或者不熟悉的用户登录，同时还可以根据用户名以及用户登录的源地址和它们正在运行的进程来判断他们是否为非法用户。

2. 锁定可疑用户

一旦发现可疑用户，就要马上将其锁定，例如上面执行“w”命令后发现 nobody 用户应该是个可疑用户（因为 nobody 默认情况下是没有登录权限的），于是首先锁定此用户，执行如下操作：

```
[root@server ~]# passwd -l nobody
```

锁定之后，有可能此用户还处于登录状态，因此还要将此用户踢下线，根据上面“w”命令的输出，即可获得此用户登录进行的 pid 值，操作如下：

```
[root@server ~]# ps -ef|grep @pts/3
531 6051 6049 0 19:23 ? 00:00:00 sshd: nobody@pts/3
[root@server ~]# kill -9 6051
```

这样就将可疑用户 nobody 从线上踢下去了。如果此用户再次试图登录，此时他已经无法登录了。

3. 通过 last 命令查看用户登录事件

last 命令记录着所有用户登录系统的日志，可以用来查找非授权用户的登录事件，而 last 命令的输出结果来源于 /var/log/wtmp 文件，稍有经验的入侵者都会删掉 /var/log/wtmp 以清除自己行踪，但还是会在此文件中露出蛛丝马迹的。

1.6.3 查看系统日志

查看系统日志是查找攻击源最好的方法，可查看的系统日志有 /var/log/messages、/var/log/secure 等，这两个日志文件可以记录软件的运行状态以及远程用户的登录状态，还可以查看每个用户目录下的 .bash_history 文件，特别是 /root 目录下的 .bash_history 文件，这个文件中记录着用户执行的所有历史命令。

1.6.4 检查并关闭系统可疑进程

检查可疑进程的命令很多，例如 ps、top 等，但是有时候只知道进程的名称无法得知路径，此时可以通过如下方式查看。

首先通过 pidof 命令查找正在运行的进程 PID，例如，要查找 sshd 进程的 PID，执行如下命令：

```
[root@server ~]# pidof sshd
13276 12942 4284
```

然后进入内存目录，查看对应 PID 目录下 exe 文件的信息：

```
[root@server ~]# ls -al /proc/13276/exe
lrwxrwxrwx 1 root root 0 Oct  4 22:09 /proc/13276/exe -> /usr/sbin/sshd
```

这样就找到了进程对应的完整执行路径。如果还有查看文件的句柄，可以查看如下目录：

```
[root@server ~]# ls -al /proc/13276/fd
```

通过这种方式基本可以找到任何进程的完整执行信息，此外还有很多类似的命令可以帮助系统运维人员查找可疑进程。例如，可以通过指定端口或者 tcp、udp 协议找到进程 PID，进而找到相关进程：

```
[root@server ~]# fuser -n tcp 111
111/tcp:          1579
[root@server ~]# fuser -n tcp 25
25/tcp:          2037
[root@server ~]# ps -ef|grep 2037
root      2037      1  0 Sep23 ?        00:00:05 /usr/libexec/postfix/master
postfix   2046    2037  0 Sep23 ?        00:00:01 qmgr -l -t fifo -u
postfix   9612    2037  0 20:34 ?        00:00:00 pickup -l -t fifo -u
root     14927 12944  0 21:11 pts/1    00:00:00 grep 2037
```

有些时候，攻击者的程序隐藏很深，例如 rootkit 后门程序，在这种情况下 ps、top、netstat 等命令也可能已经被替换，如果再通过系统自身的命令去检查可疑进程就变得毫不可信，此时，就需要借助于第三方工具来检查系统可疑程序，例如前面介绍过的 chkrootkit、RKHunter 等工具，通过这些工具可以很方便地发现系统被替换或篡改的程序。

1.6.5 检查文件系统的完好性

检查文件属性是否发生变化是验证文件系统完好性最简单、最直接的方法，例如，可以检查被入侵服务器上 /bin/ls 文件的大小是否与正常系统上此文件的大小相同，以验证文件是否被替换，但是这种方法比较低级。此时可以借助于 Linux 下的 RPM 工具来完成验证，操作如下：

```
[root@server ~]# rpm -Va
....L... c /etc/pam.d/system-auth
S.5..... c /etc/security/limits.conf
S.5....T c /etc/sysctl.conf
S.5....T  /etc/sgml/docbook-simple.cat
```



```

S.5....T c /etc/login.defs
S.5..... c /etc/openldap/ldap.conf
S.5....T c /etc/sudoers
..5....T c /usr/lib64/security/classpath.security
....L... c /etc/pam.d/system-auth
S.5..... c /etc/security/limits.conf
S.5..... c /etc/ldap.conf
S.5....T c /etc/ssh/sshd_config

```

对于输出中每个标记的含义介绍如下。

- S 表示文件长度发生了变化。
- M 表示文件的访问权限或文件类型发生了变化。
- 5 表示 MD5 校验和发生了变化。
- D 表示设备节点的属性发生了变化。
- L 表示文件的符号链接发生了变化。
- U 表示文件 / 子目录 / 设备节点的 owner 发生了变化。
- G 表示文件 / 子目录 / 设备节点的 group 发生了变化。
- T 表示文件最后一次的修改时间发生了变化。

如果在输出结果中有“M”标记出现,那么对应的文件可能已经遭到篡改或替换,此时可以通过卸载这个 RPM 包重新安装来清除受攻击的文件。

不过这个命令有个局限性,那就是只能检查通过 RPM 包方式安装的所有文件,对于通过非 RPM 包方式安装的文件就无能为力了。同时,如果 RPM 工具也遭到替换,就不能使用这种方法了,此时可以从正常的系统上复制一个 RPM 工具进行检测。当然,对文件系统的检查也可以通过 chkrootkit、RKHunter 这两个工具来完成,上面介绍的命令或工具可以作为辅助或者补充。

1.7 一次 Linux 被入侵后的分析

下面通过一个案例来介绍一个服务器被 rootkit 入侵后的处理思路和处理过程。rootkit 攻击是 Linux 系统下最常见的攻击手段和攻击方式。

1.7.1 受攻击现象

这是一台客户的门户网站服务器,托管在电信机房。客户接到电信的通知:由于此服务器持续对外发送数据包,导致 100MB 带宽耗尽,于是电信切断了此服务器的网络。此服务器安装了 CentOS 5.5 版本的系统,对外开放了 80、22 端口。

从客户那里了解到,网站的访问量并不大,所以带宽占用也不会太高,而耗尽 100MB

的带宽是绝对不可能的，那么极有可能是服务器遭受了流量攻击，于是登录服务器做详细的检测。

1.7.2 初步分析

在电信人员的配合下通过交换机对该服务器的网络流量进行了检测，发现该主机确实存在对外 80 端口的扫描流量，于是登录系统通过 `netstat -an` 命令对系统开启的端口进行检查。奇怪的是，没有发现任何与 80 端口相关的网络连接。接着使用 `ps -ef`、`top` 等命令也没有发现任何可疑的进程。于是怀疑系统可能被植入了 rootkit。

为了验证系统是否被植入了 rootkit，我们将网站服务器下的 `ps`、`top` 等命令与之前备份的同版本可信操作系统命令进行 `md5sum` 校验，结果发现网站服务器下的这两个命令确实被修改过，由此断定，此服务器已经被入侵、且安装了 rootkit 级别的后门程序。

1.7.3 断网分析系统

由于服务器不停向外发包，因此，首先要做的就是将此服务器断开网络，然后分析系统日志，寻找攻击源。但是系统命令已经被替换，如果继续在该系统上执行操作将变得不可信。这里可以通过两种方法来避免这种情况，第一种方法是从此服务器的硬盘取下来挂载到另外一台安全的主机上进行分析，另一种方法是从一个同版本可信操作系统下复制所有命令到这台入侵服务器下某个路径，然后在执行命令的时候指定此命令的完整路径即可。这里采用第二种方法。

首先查看系统的登录日志，查看是否有可疑登录信息，执行如下命令：

```
more /var/log/secure |grep Accepted
```

通过查看命令输出，有一条日志引起了我们的怀疑：

```
Oct 3 03:10:25 webserver sshd[20701]: Accepted password for mail from 62.17.163.186
port 53349 ssh2
```

这条日志显示在 10 月 3 日凌晨 3 点 10 分，有个 mail 账号从 62.17.163.186 这个 IP 成功登录了系统，mail 是系统的内置账号，默认无法执行登录操作的，而经过查证 62.17.163.186 这个 IP，是来自爱尔兰的一个地址。从 mail 账号登录的时间来看，早于此网站服务器遭受攻击的时间。

接着查看系统密码文件 `/etc/shadow`，又发现可疑信息：

```
mail:$1$kCEd3yD6$W1evaY5BMPQIqfTwTVJiX1:15400:0:99999:7:::
```

很明显，mail 账号已经被设置了密码，并且被修改为可远程登录，之所以使用 mail 账

号,猜想可能是因为入侵者想留下一个隐蔽的账号,以方便日后再次登录系统。

然后继续查看其他系统日志,如 /var/log/messages、/var/log/wtmp 等均为空文件,可见,入侵者已经清理了系统日志文件,至于为何没有清空 /var/log/secure 文件,就不得而知了。

1.7.4 寻找攻击源

到目前为止,我们所知道的情况是,有个 mail 账号曾经登录过系统,但是为何会导致此网站服务器持续对外发送数据包呢?必须找到对应的攻击源,通过替换到此服务器上的 ps 命令查看系统目前运行的进程,又发现了新的可疑:

```
nobody 22765 1 6 Sep29 ? 4-00:11:58 .t
```

这个 .t 程序是什么呢?继续执行 top 命令,结果如下:

```
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
22765 nobody 150 1740m 1362m 1228 S 98.3 91.5 2892:19 .t
```

从输出可知,这个 t 程序已经运行了 4 天左右,运行这个程序的是 nobody 用户,并且这个 t 程序消耗了大量的内存和 CPU,这也是之前客户反映的网站服务器异常缓慢的原因。根据这个输出,我们得到了 t 程序的进程 PID 为 22765。接下来根据 PID 查找下执行程序的路径。

进入内存目录,查看对应 PID 目录下 exe 文件的信息:

```
[root@webserver ~]# /mnt/bin/ls -al /proc/22765/exe
lrwxrwxrwx 1 root root 0 Sep2922:09 /proc/22765/exe -> /var/tmp/.../apa/t
```

这样就找到了进程对应的完整程序执行路径,这个路径很隐蔽,由于 /var/tmp 目录在默认情况下任何用户可读,而入侵者就是利用这个漏洞在 /var/tmp 目录下创建了一个“...”的目录,而在这个目录下隐藏着攻击的程序源。进入 /var/tmp/.../ 目录,发现了入侵者放置的一系列 rootkit 文件,列表如下:

```
[root@webserver ...]# /mnt/bin/ls -al
drwxr-xr-x 2 nobody nobody 4096 Sep 29 22:09 apa
-rw-r--r-- 1 nobody nobody 0 Sep 29 22:09 apa.tgz
drwxr-xr-x 2 nobody nobody 4096 Sep 29 22:09 caca
drwxr-xr-x 2 nobody nobody 4096 Sep 29 22:09 haha
-rw-r--r-- 1 nobody nobody 0 Sep 29 22:10 kk.tar.gz
-rwxr-xr-x 1 nobody nobody 0 Sep 29 22:10 login
-rw-r--r-- 1 nobody nobody 0 Sep 29 22:10 login.tgz
-rwxr-xr-x 1 nobody nobody 0 Sep 29 22:10 z
```

通过分析这些文件,基本断定这就是我们要找的程序攻击源,其中:

1) z 程序是用来清除系统日志等相关信息的, 例如执行:

```
./z 62.17.163.186
```

在执行这条命令后, 系统中所有与 62.17.163.186 有关的日志将全部被清除。

2) 在 apa 目录下有个后门程序 t, 这就是之前在系统中看到的程序。在运行此程序后, 此程序会自动读 apa 目录下的 ip 这个文件, 而 ip 这个文件记录了各种 ip 地址信息, 猜想这个 t 程序应该是去扫描 ip 文件中记录的所有 ip 信息, 进而获取远程主机的权限。可见这个网站服务器已经是入侵者的一个肉鸡了。

3) haha 目录下放置的就是用来替换系统相关命令的程序, 也就是说这个目录下的程序使我们无法看到操作系统的异常情况。

4) login 程序就是用来替换系统登录程序的木马程序, 此程序还可以记录登录账号和密码。

1.7.5 查找攻击原因

到这里为止, 服务器上遭受的攻击已经基本清晰了, 但是入侵者是如何侵入这台服务器的呢? 这个问题很重要, 一定要找到入侵的根源, 才能从根本上封堵漏洞。

为了弄清楚入侵者是如何进入服务器的, 需要了解此服务器的软件环境, 这台服务器是一台基于 Java 的 Web 服务器, 安装的软件有 Apache2.0.63、Tomcat5.5, Apache 和 Tomcat 之间通过 mod_jk 模块进行集成, Aapache 对外开放 80 端口。由于 Tomcat 没有对外开放端口, 因此将问题集中到 Apache 上面。

通过查看 Apache 的配置发现, Apache 仅仅处理些静态资源请求, 而网页也以静态页面居多, 所以通过网页方式入侵系统可能性不大。既然漏洞可能来自于 Apache, 那么尝试查看 Apache 日志, 也许能发现一些可疑的访问痕迹。通过查看 access.log 文件, 发现了如下信息:

```
62.17.163.186 - - [29/Sep/2013:22:17:06 +0800] "GET http://www.xxx.com/cgi-bin/awstats.pl?configdir=/echo;echo;ps+-aux%00 HTTP/1.0" 200 12333 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; pt-BR; rv:1.8.1) Gecko/20121010 Firefox/2.0"
```

```
62.17.163.186 - - [29/Sep/213:22:17:35 +0800] "GET http://www.xxx.com/cgi-bin/awstats.pl?configdir=/echo;echo;cd+/var/tmp/.../haha;ls+-a%00 HTTP/1.0" 200 1626 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; pt-BR; rv:1.8.1) Gecko/20121010 Firefox/2.0"
```

至此, 发现了漏洞的根源, 原来是 awstats.pl 脚本中 configdir 的一个漏洞, 通过了解此服务器的应用, 客户确实是通过一个 Awstats 的开源插件来做网页访问统计, 通过这个漏

洞, 攻击者可以直接在浏览器上操作服务器, 例如查看进程、创建目录等。通过上面第二条日志可以看出, 攻击者将正常浏览器执行切换到 /var/tmp/.../haha 目录的操作。

这个脚本漏洞挺可怕的, 不过在 Awstats 官网上也早已给出了修补的方法。对于这个漏洞, 修复方法很简单, 打开 awstats.pl 文件, 找到如下信息:

```
if ($QueryString =~ /configdir=([^\&]+)/i)
{
    $DirConfig=&DecodeEncodedString("$1");
}
```

修改为如下即可:

```
if ($QueryString =~ /configdir=([^\&]+)/i)
{
    $DirConfig=&DecodeEncodedString("$1");
    $DirConfig=~tr/a-z0-9_\-\./\./a-z0-9_\-\./\./cd;
}
```

1.7.6 揭开谜团

通过前面的逐步分析和介绍, 对此服务器遭受入侵的原因和过程已经非常清楚了, 大致过程如下:

1) 攻击者通过 Awstats 脚本 awstats.pl 文件的漏洞进入系统, 在 /var/tmp 目录下创建了隐藏目录, 然后将 rootkit 后门文件传到这个路径下。

2) 攻击者通过植入后门程序, 获取了系统超级用户权限, 进而控制了这台服务器, 通过这台服务器向外发包。

3) 攻击者的 IP 地址 62.17.163.186 可能是通过代理过来的, 也可能是攻击者控制的其他肉鸡服务器。

4) 攻击者为了永久控制这台机器, 修改了系统默认账号 mail 的信息, 将 mail 账号变为可登录, 并且设置了 mail 账号的密码。

5) 攻击者在完成攻击后, 通过后门程序自动清理了系统访问日志, 毁灭了证据。

通过对这个入侵过程的分析, 发现入侵者的手段还是非常简单和普遍的, 虽然入侵者删除了系统的一些日志, 但还是留下了很多可查的踪迹。其实还可以查看用户下的 .bash_history 文件, 这个文件是用户操作命令的历史记录。

1.7.7 如何恢复网站

由于系统已经文件被更改和替换, 此系统已经变得完全不可信, 因此建议备份网站数

Linux 网络安全运维

2.1 网络实时流量监测工具 iftop

网络管理是基础运维中一项很重要的工作，在看似平静的网络运行中，其实暗流汹涌，要保证业务系统稳定运行，网络运维人员必须了解网络的流量状态、各个网段的使用情形，带宽的利用率、网络是否存在瓶颈等。同时，当网络发生故障时，必须能够及时发现问题，迅速定位问题，进而解决问题，这就需要一些网络监测工具的辅助，本节将介绍一款小巧但功能很强大的网络实时流量监测工具 iftop。

2.1.1 iftop 能做什么

iftop 是一个免费的网卡实时流量监控工具，类似于 Linux 下的 top 命令。iftop 可以监控指定网卡的实时流量、端口连接信息、反向解析 IP 等，还可以精确显示本机网络流量情况及网络内各主机与本机相互通信的流量集合，非常适合于监控代理服务器或路由器的网络流量。同时，iftop 对检测流量异常的主机非常有效，通过 iftop 的输出可以迅速定位主机流量异常的根源，这对于网络故障排查、网络安全检测是十分有用的。

2.1.2 iftop 的安装

iftop 的官方网站为 <http://www.ex-parrot.com/pdw/iftop/>，目前最新稳定版本为 iftop-0.17。安装 iftop 非常简单，有源码编译安装和 yum 源方式安装两种方式，这里以 CentOS 6.4 版本为例，简单介绍如下。

(1) 源码编译安装 iftop

安装 iftop 必需的软件库：

```
[root@localhost ~]# yum install libpcap libpcap-devel ncurses ncurses-devel
[root@localhost ~]# yum install flex byacc
```

下载 iftop，编译安装：

```
[root@localhost ~]# wget http://www.ex-parrot.com/pdw/iftop/download/iftop-0.17.tar.gz
[root@localhost ~]# tar zxvf iftop-0.17.tar.gz
[root@localhost ~]# cd iftop-0.17
[root@localhost ~]# ./configure
[root@localhost ~]# make
[root@localhost ~]# make install
```

(2) yum 源方式安装

安装 iftop 必需的软件库：

```
[root@localhost ~]# yum install libpcap libpcap-devel ncurses ncurses-devel
[root@localhost ~]# yum install flex byacc
[root@localhost ~]# wget http://dl.fedoraproject.org/pub/epel/6/i386/epel-release-6-8.noarch.rpm
[root@localhost ~]# rpm -ivh epel-release-6-8.noarch.rpm
[root@localhost ~]# yum install iftop
```

这样，iftop 就安装完成了。

2.1.3 使用 iftop 监控网卡实时流量

安装完 iftop 工具后，直接输入 iftop 命令即可显示网卡实时流量信息。在默认情况下，iftop 显示系统第一块网卡的流量信息，如果要显示指定网卡信息，可通过“-i”参数实现。

(1) iftop 输出界面说明

执行“iftop -P -i em1”命令，得到如图 2-1 所示的 iftop 的一个典型输出界面。

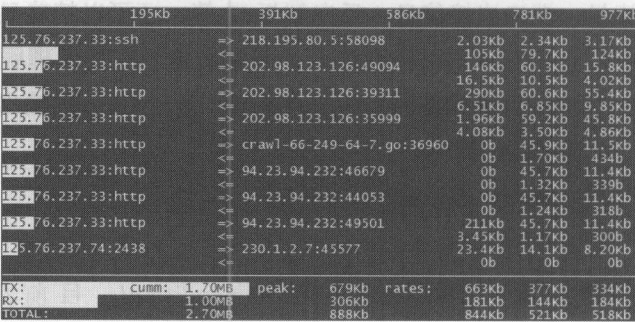


图 2-1 iftop 状态监控界面

iftop 的输出从整体上可以分为三大部分。

- 第一部分是 iftop 输出中最上面的一行，此行信息是流量刻度，用于显示网卡带宽流量。
- 第二部分是 iftop 输出中最大的一个部分，此部分又分为左、中、右三列，左列和中列记录了哪些 IP 或主机正在和本机的网络进行连接。其中，中列的“=>”代表发送数据，“<=”代表接收数据，通过这个指示箭头可以很清晰地知道两个 IP 之间的通信情况。最右列又分为三小列，这些实时参数分别表示外部 IP 连接到本机 2s、10s 和 40s 内的平均流量值。另外，这个部分还有一个流量图形条，流量图形条是对流量大小的动态展示，以第一部分中的流量刻度为基准。通过这个流量图形条可以很方便地看出哪个 IP 的流量最大，进而迅速定位网络中可能出现的流量问题。
- 第三部分位于 iftop 输出的最下面，可以分为三行，其中，“TX”表示发送数据，“RX”表示接收数据，“TOTAL”表示发送和接收的全部流量。与这三行对应的有三列，其中，“cum”列表示从运行 iftop 到目前的发送、接收和总数据流量；“peak”列表示发送、接收以及总的流量峰值；“rates”列表示过去 2s、10s、40s 内的平均流量值。

(2) iftop 使用参数说明

iftop 还有很多附加参数和功能。执行“iftop -h”命令即可显示 iftop 可使用的所有参数信息。iftop 常用的参数以及含义如表 2-1 所示。

表 2-1 iftop 常用参数说明

参数	含义	示例
-i	指定需要监测的网卡	iftop -i em1
-n	将输出的主机信息都通过 IP 显示，不进行 DNS 反向解析	iftop -n
-B	将输出以 byte 为单位显示网卡流量，默认是 bit	iftop -B
-p	以混杂模式运行 iftop，此时 iftop 可以用作网络嗅探器	iftop -p
-N	只显示连接端口号，不显示端口对应的服务名称	iftop -N
-P	显示主机以及端口信息，这个参数非常有用	iftop -P
-F	显示特定网段的网卡进出流量	iftop -F 192.168.12.0/24
-m	设置 iftop 输出界面中最上面的流量刻度最大值，流量刻度分 5 个大段显示	iftop -m

(3) iftop 的交互操作

在 iftop 的实时监控界面中，还可以对输出结果进行交互式操作，用于对输出信息进行整理和过滤，在图 2-1 所示界面中，按“h”键即可进入交互选项界面，如图 2-2 所示。

iftop 的交互功能和 Linux 下的 top 命令非常类似，交互参数主要分为四个部分，分别是一般参数、主机显示参数、端口显示参数和输出排序参数。相关参数的含义如表 2-2 所示。

```

1.91Mb      3.81Mb      5.72Mb      7.63Mb      9.54Mb
Host display:
n - toggle DNS host resolution
s - toggle show source host
d - toggle show destination host
t - cycle line display mode

Port display:
N - toggle service resolution
S - toggle show source port
D - toggle show destination port
p - toggle port display

Sorting:
1/2/3 - sort by 1st/2nd/3rd column
< - sort by source name
> - sort by dest name
o - freeze current order

General:
P - pause display
h - toggle this help display
b - toggle bar graph display
B - cycle bar graph average
T - toggle cumulative line totals
j/k - scroll display
f - edit filter code
l - set screen filter
L - lin/log scales
! - shell command
q - quit

TX:          cumm: 76.1MB   peak: 992kb   rates: 688b   2.36kb   107kb
RX:          13.0MB   112kb     13.7kb   9.67kb   21.5kb
TOTAL:       89.1MB   1.03Mb    14.3kb   12.0kb   128kb
```

图 2-2 iftop 的交互选项界面

表 2-2 iftop 交互参数说明

参数	含义
P	通过此键可切换暂停 / 继续显示
h	通过此键可在交互参数界面 / 状态输出界面之间来回切换
b	通过此键可切换是否显示平均流量图形条
B	通过此键可切换显示 2s、10s、40s 内的平均流量
T	通过此键可切换是否显示每个连接的总流量
j/k	按“j”键或“k”键可以向上或向下滚动屏幕显示当前的连接信息
l	通过此键可打开 iftop 输出过滤功能，比如输入要显示的 IP，按回车键后，屏幕就只显示与这个 IP 相关的流量信息
L	通过此键可切换显示流量刻度范围，刻度不同，流量图形条也会不同
q	通过此键可退出 iftop 流量监控界面
n	通过此键可使 iftop 输出结果以 IP 或主机名的方式显示
s	通过此键可切换是否显示源主机信息
d	通过此键可切换是否显示远端目标主机信息
t	通过此键可切换 iftop 显示格式，连续按此键可依次显示：以两行方式显示发送、接收流量，以一行方式显示发送、接收流量，只显示发送流量 / 只显示接收流量
N	通过此键可切换显示端口号 / 端口号对应服务名称
S	通过此键可切换是否显示本地源主机的端口信息
D	通过此键可切换是否显示远端目标主机的端口信息
p	通过此键可切换是否显示端口信息
<	通过此键可根据左边的本地主机名或 IP 地址进行排序
>	通过此键可根据远端目标主机的主机名或 IP 地址进行排序
o	通过此键可切换是否固定显示当前的连接

iftop 的强大之处在于它能够实时显示网络的流量状态，监控网卡流量的来源 IP 和目标地址，这对于检测服务器网络故障、流量异常是非常有用的，只需通过一个命令就能把流量异常或网络故障的原因迅速定位，因此，对于运维人员来说，iftop 是必不可少的一个网络故障排查工具。

2.2 网络流量监控与分析工具 Ntop 和 Ntopng

对于单台服务器网络故障的排查，iftop 工具可以轻松实现，但是在监控一个庞大的服务器网络，并且要分析每台主机以及端口的网络状态时，iftop 就显得爱莫能助了，这个时候就需要一个高效的网络管理系统了。Ntop 就是一个功能强大的流量监控、端口监控、服务监控管理系统。

2.2.1 Ntop 与 MRTG 的异同

对于 MRTG，读者可能并不陌生，它是一个监控网络链路流量的工具，通过 SNMP 协议得到设备的流量信息，并将信息通过图形展示给用户。MRTG 配置简单，容易使用，它的优点是耗用的系统资源小，可以非常直观地显示流量负载，但是它也有很多缺点，例如：只能用于 TCP/IP 网络、数据不能重复使用、无法记录更详细的流量状态、没有管理功能等，而这些刚好是 Ntop 最擅长的地方。

Ntop 是网络流量监控中的新贵，它是一种网络嗅探器，在监测网络数据传输、排除网络故障方面功能十分强大。它通过分析网络流量来判断网络上存在的各种问题，还可以监控是否有黑客正在攻击网络，如果网络突然变缓慢，通过 Ntop 截获的数据包，可以确定是什么类型的数据包占据了大量带宽，以及数据包的发送时间、数据包传送的延时、数据包的来源地址等，通过这些信息，运维人员可以及时做出响应，或者对网络进行调整，从而保证网络正常、稳定运行。

2.2.2 Ntop 与 Ntopng 的功能介绍

Ntop 提供了命令行界面和 Web 界面两种工作方式，通过 Web 界面，可以清晰展示网络的整体使用情况、网络中各主机的流量状态与排名、各主机占用的带宽以及各时段的流量明细、局域网内各主机的路由、端口使用情况等。

根据官方的介绍，Ntop 主要提供以下几个功能：

- 可以自动从网络中获取有用的信息。
- 可以将获取的数据包信息转换为可识别的格式。
- 可以记录网络的通信时间和过程。
- 可以对网络中失败的通信进行分析。
- 可以发现网络环境中通信的瓶颈。
- 可以自动识别客户端正在使用的操作系统。

通过对 Ntop 功能的介绍，不难看出，它就是从分析网络流量角度来确定网络上存在的

各种问题,说得更简单一点,它就是一个抓包工具,然后通过归纳和绘图实现更多的功能。

在 Ntop 版本更新到 Ntop5.x 之后,官方宣布停止 Ntop 版本的更新,继而推出替代版本 Ntopng。Ntopng 在 Ntop 版本的基础上,去掉了一些拖沓冗长的功能,同时新增了网络流量实时监控功能,并将各个功能进行重新梳理和整合,使整个流量展示更加智能化和合理化。

Ntopng 使用 Redis 键值服务按时间序列存储统计信息,通过这种方式实现了流量状态实时展示。与 Ntop 类似,Ntopng 也内置 Web 服务功能,同时,也支持命令行界面和 Web 界面两种工作方式,但是 Ntopng 降低了对 CPU 和内存的使用率,资源消耗更少。Ntopng 除了可以实现 Ntop 的所有功能外,新增的功能如下:

- ❑ 以图形的方式动态展示流量状态。
- ❑ 实时监控网络数据并实时汇总。
- ❑ 以矩阵图的方式显示 IP 流量。
- ❑ 可以生成基于 HTML5/AJAX 的网络流量统计。
- ❑ 支持历史流量数据分析。
- ❑ 基于 HTML5 的动态图形用户界面。

下面分别介绍 Ntop 和 Ntopng 的安装及使用技巧。

2.2.3 安装 Ntop 与 Ntopng

1. 安装 Ntop

Ntop 支持 Win32、Linux、UNIX、BSD 等平台。可以在 Ntop 官方网站 <http://www.ntop.org/> 下载对应的版本。Ntop 的安装可以通过 yum 源方式和源码编译安装两种方式实现,为了能够使用最新的稳定版本,这里采用源码编译的方式来安装,安装过程如下。

(1) 安装 Ntop 必需的软件包

这里安装的操作系统环境为 CentOS 6.3 x86_64 版本。为了顺利完成源码编译,需要安装 Ntop 必需的一些软件包,操作如下:

```
[root@monitor ~]# yum -y install libpcap libpcap-devel libtool libpng gdbm gdbm-
devel glib libxml2-devel pango pango-devel gd zlib zlib-devel
[root@monitor ~]# yum -y install svn rrdtool rrdtool-devel python python-devel
GeoIP GeoIP-devel
```

(2) 编译安装 Ntop

这里下载的是 ntop-5.0.1 版本,编译安装过程如下:

```
[root@monitor ~]# tar zxvf ntop-5.0.1.tar.gz
[root@monitor ~]# cd ntop-5.0.1
```



```
[root@monitor ~]# ./autogen.sh --with-tcpwrap
[root@monitor ~]# make
[root@monitor ~]# make install
```

其中, `--with-tcpwrap` 选项用于支持 `tcp_wrappers` 访问控制, 以保证 Ntop Web 访问的安全。

(3) 简单配置 Ntop

在 Ntop 安装完成后, 默认的数据存放目录为 `/usr/local/var/ntop`。为了保证安全, 建议以低权限用户 `nobody` 身份运行 Ntop 进程, 因此, 可能需要对 Ntop 默认的数据存放目录进行权限调整, 执行如下命令即可:

```
[root@monitor ~]# chown -R nobody /usr/local/var/ntop
```

通过 Ntop 的 Web 页面可以修改 Ntop 的设置或关闭 Ntop 服务, 但是必须通过管理员用户的验证, Ntop 默认的管理员为 `admin`, 密码为空, 因此, 需要为其设置一个密码。通过如下命令即可设置 `admin` 用户的密码:

```
[root@monitor ~]# ntop -A
```

然后重复输入两次密码即可。

Ntop 的 Web 页面在默认情况下没有访问限制, 有时候为了网络的安全, 建议设置授权访问, 只有授权的主机才能访问此 Web 页面, 这可以通过 Linux 系统本身的 `tcp_wrappers` 功能实现, 授权过程如下:

```
[root@monitor ~]# vim /etc/hosts.allow
ntop: 192.168.12.188
[root@monitor ~]# vim /etc/hosts.deny
ntop: ALL
```

这里设置只允许 IP 地址为 192.168.12.188 的主机可以访问 Ntop 的 `ntop` 服务, 禁止其他所有 IP 访问。

2. 安装 Ntopng

Ntopng 是目前 Ntop 官方的主推版本, 可以从 <http://www.ntop.org/> 下载目前最新的 `ntopng-1.1` 源码版本进行编译安装。不过为了安装方便, 官方推出了 Ntopng 的 yum 源仓库, 通过 yum 源仓库可以轻松安装 Ntopng, 这里就采用 yum 源方式进行安装。

(1) 设置 yum 源

首先为 Ntopng 创建一个 yum 源仓库, 内容如下:

```
[root@localhost ~]# cat /etc/yum.repos.d/ntop.repo
[ntop]
```

```
name=ntop packages
baseurl=http://www.nmon.net/centos/$releasever/$basearch/
enabled=1
gpgcheck=1
gpgkey=http://www.nmon.net/centos/RPM-GPG-KEY-deri
```

然后下载一个 epel 的 yum 源:

```
[root@localhost ~]# wget \
> http://download.fedoraproject.org/pub/epel/6/x86_64/epel-release-6-8.noarch.rpm
[root@localhost ~]# rpm -Uvh epel-release-6-8.noarch.rpm
```

(2) 安装 Ntopng

在设置好两个 yum 源后, 安装 Ntopng 就变得十分简单了, 只需执行如下操作即可:

```
[root@localhost ~]# yum clean all
[root@localhost ~]# yum update
[root@localhost ~]# yum install pfring n2disk nProbe ntopng ntopng-data nbox
```

(3) 配置 Ntopng

在 Ntopng 安装完成后, 默认的配置文件的模板是 /etc/ntopng/ntopng.conf.sample, 可以将此文件重命名为 ntopng.conf, 然后在这个配置文件中添加一些配置信息, 例如:

```
[root@localhost ~]# cat /etc/ntopng/ntopng.conf
-G=/var/tmp/ntopng.gid
--local-networks "192.168.12.0/24"
--interface em2
--user nobody
--http-port 3000
```

相关参数含义如下:

- ❑ -G 指定存储 Ntopng 进程号的文件路径。
- ❑ --local-network 指定要监控的本地子网段。
- ❑ --interface em2 指定监听 em2 网卡上的流量。
- ❑ --user 指定运行 Ntopng 服务所使用的账户。
- ❑ --http-port 指定 Ntopng 的 Web 服务端口号, 如果不指定, 默认端口为 3000。

(4) 启动 Ntopng 服务

在启动 Ntopng 服务之前, 需要先启动 redis 服务。redis 的功能之前介绍过, 主要为 Ntopng 提供键值存储。下面首先启动 redis 服务, 然后启动 Ntopng 服务, 执行操作如下:

```
[root@localhost ~]# service redis start
[root@localhost ~]# service ntopng start
```

为了保证 redis 和 Ntopng 服务在以后可开机自启动，还需要执行如下操作：

```
[root@localhost ~]# chkconfig ntopng on
[root@localhost ~]# chkconfig redis on
```

最后，就可以通过 Web 方式（http://IP:3000）来访问 Ntopng 提供的服务了，默认登录用户名和密码均为 admin，可在登录后进行修改。

2.2.4 Ntop 和 Ntopng 的使用技巧

在完成 Ntop 安装后，执行如下命令即可启动 Ntop 服务：

```
[root@networkserver ~]# ntop -i em1 -L -d
```

这里通过 Ntop 命令监控网卡 em1 的流量状态，相关参数的含义将在后面章节详细介绍。在执行此命令后，Ntop 服务的日志输出将重定向到系统的 /var/log/messages 文件中，同时将开启默认的 3000 端口作为 Web 界面服务端口，执行 http://IP:3000 即可访问 Ntop 提供的 Web 监控界面。

1. Web 界面下 Ntop 的使用方法与技巧

Ntop 的 Web 界面主要由 7 个主栏目组成，下面介绍每个栏目中需要重点关注的功能点。

“About” 栏目包括 Ntop 的简单介绍和一些在线手册等帮助信息。

“Summary” 栏目是对目前网络流量的一个整体概况，其中子栏目“Traffic”可以显示全局流量统计，主要包含网络接口流量统计、协议流量分布、应用协议流量统计等，网络流量会以柱面图、曲线图和明细表格的形式展示。图 2-3 显示的是 L2/L3 协议对应的流量分布图。

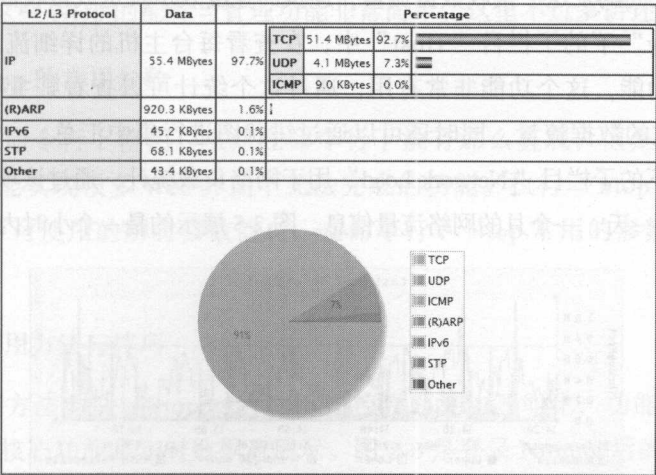


图 2-3 Ntop 根据协议进行的流量分布统计

“Summary”下的子栏目“Host”主要显示所有可监控主机的IP地址、地理位置、MAC地址、数据发送接收量、目前活动连接数等各种信息，在主机流量监控方面，可通过Bytes方式统计，也可以用packets方式统计，要了解每个主机的详细流量信息，只需单击对应的Host便可查看，图2-4就是某主机在某时刻的流量连接流视图。

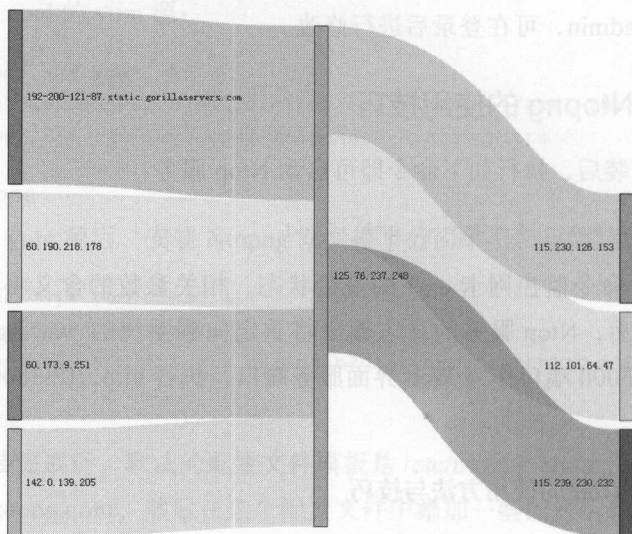


图 2-4 通过 Ntop 展示某主机在某时刻的连接流视图

通过图2-4可以非常清晰地了解某主机在某时刻的连接状态，中间的竖柱表示的是IP为“125.76.237.248”的这台主机，然后以竖柱为中心，分成左右两个部分，左边部分表示外部IP与“125.76.237.248”这台主机之间的发送、接收数据流量，右边部分表示“125.76.237.248”这台主机与外部IP之间的发送、接收数据流量，连接的宽度表示发送或接收数据量的大小。

在“Summary”下的子栏目“Host”中，在查看每台主机的详细流量页面上有一个按时段的流量统计功能，这个功能非常有用，通过这个统计可以查看某主机在一天任意一个小时内发送、接收的数据流量，同时还可以通过饼状图进行集中汇总。

“Summary”下的子栏目“Network Load”用于网络负载统计，通过该项功能可以查看最近10分钟、一小时、一天、一个月的网络流量信息。图2-5展示的是一个小时内的网络负载统计。

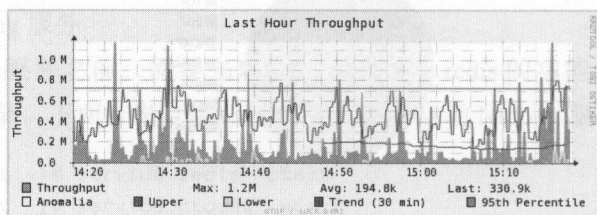


图 2-5 Ntop 展示的一个小时内的网络负载

“All Protocols”栏目主要用于查看各主机发送、接收的数据量，并将数据以 TCP、UDP、ICMP 的方式进行分类统计。其中，子栏目“Throughput”主要显示所有可见主机的吞吐量，子栏目“Activity”主要显示当前网络可见主机在 24 小时中每小时的流量状态，并且每个时段根据流量的大小分别用不同的颜色进行标注。

“IP”栏目主要对各台主机中应用层协议产生的流量进行统计。例如，子栏目“Summary”主要对各主机中 HTTP、FTP、Mail、SSH、DNS 等服务产生的流量进行详细统计，同时还可以统计多播信息、流量分布等；子栏目“Traffic Directions”主要用于统计端到端的流量信息，可以统计本地到本地、本地到远端、远端到本地、远端到远端的流量状态；子栏目“Local”主要是统计局域网内各主机使用状况，比如可以统计本地路由使用信息、本地端口使用信息、Active Sessions 连接信息等。

“Utils”栏目主要包括 RRD 参数的配置、转存 Ntop 的统计信息，以及查看 Ntop 运行日志信息等功能。

“Plugins”栏目用于继承 Ntop 插件工具，默认安装的插件有 NetFlow、rrdPlugin、sFlow 等，其中，NetFlow 插件可用于设置、激活、停用 NetFlow 支持，在启用 NetFlow 后，Ntop 就可以统计 NetFlow 的详细信息，包括 NetFlow 的格式、数据量及端口流量。而 rrdPlugin 插件主要用于生成流量图，它比 MRTG 更灵活，非常适合用 shell、perl 等程序来调用，以生成所需的图片。sFlow 是一种新的网络监测技术，可适应超大网络流量下的流量数据分析，在 Ntop 中启用 sFlow 支持后，不但可以降低实施成本，也可以解决网络管理中面临的很多问题。

最后一个栏目“admin”是一个管理选项，访问此栏目时需要提供管理员密码，有 Ntop 的参数配置、登录 Ntop 的密码设置、配置用户访问 Ntop 的页面、Ntop 的启动与关闭等几个功能选项。这些 Ntop 的配置与管理功能非常简单，这里不过多讲述。

2. 命令行下 Ntop 的常用参数

Ntop 也可以在命令行下使用。虽然在命令行下没有那么直观，但是添加和修改配置非常迅速，并且还能实现很多 Web 界面下无法完成的功能。执行“ntop -h”命令即可显示 Ntop 在命令行下可使用的所有参数信息。在命令行下 Ntop 常用的参数及含义如表 2-3 所示。

3. Ntopng 的使用方法与技巧

与 Ntop 的使用方法类似，Ntopng 的 Web 监控界面更加智能化，功能展示更加统一和人性化。Ntopng 的核心功能是实时数据流展示。图 2-6 是登录 Ntopng 后的一个主界面，中间的“Top Flow Talkers”部分就是流量实时展示界面。

表 2-3 命令行下 Ntop 常用参数含义说明

参数	含义
-d	将 Ntop 进程放到后台执行，默认 Ntop 在前台运行，在屏幕输出日志
-u	指定启动 Ntop 执行的用户，默认是 nobody 用户
-i	指定 Ntop 监听的网卡设备，指定多块网卡时，用逗号隔开
-M	如果通过“-i”参数指定了多块网卡，那么输出的网卡流量信息默认是合并的，如果要将多块网卡信息分开统计，就需要添加此参数
-L	将 Ntop 的输出信息写入系统日志文件中，对应于 CentOS，就是 /var/log/messages 文件
-w	设置 Ntop 的 Web 界面使用的端口，默认是 3000
-r	设定 Ntop 的 Web 界面自动刷新的频率，默认是每 3s 刷新一次

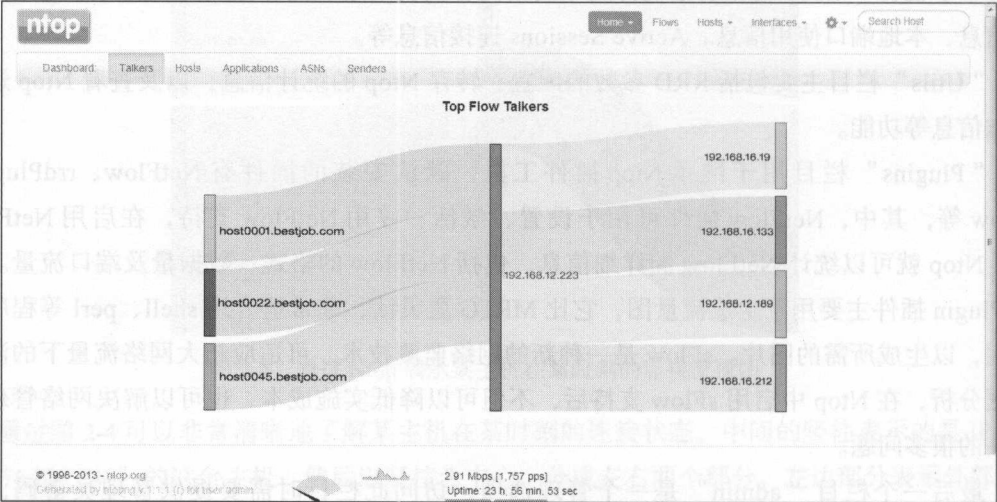


图 2-6 Ntopng 流量实时监控主界面

从图 2-6 可以看出，Ntopng 的 Web 界面主要分为 Home、Flows、Hosts 和 Interfaces 四个主栏目。其中“Home”栏目主要是从整体上展示并统计发送、接收的数据流；“Flows”栏目是基于 DPI 的自动程序或服务探测程序生成的实时数据报告，主要用于统计活跃的数据流，并将数据流以协议类型、应用类型、数据量大小等方式进行详细统计，如图 2-7 所示。

Active Flows								
Info	Application	L4 Proto	Client	Server	Duration	Breakdown	Throughput	Total Bytes
	HTTP	TCP	192.168.11.218:58375	192.168.12.223:80	2 sec		0 bps	15.59 KB
	HTTP	TCP	192.168.11.218:58374	192.168.12.223:8080	1 sec		0 bps	12.1 KB
	HTTP	TCP	192.168.11.218:58373	192.168.12.223:8080	1 sec		0 bps	10.01 KB
	HTTP	TCP	192.168.11.218:58387	192.168.12.223:80	2 sec		0 bps	8.67 KB
	HTTP	TCP	192.168.11.218:58381	192.168.12.223:80	2 sec		0 bps	8.54 KB
	HTTP	TCP	192.168.11.218:58384	192.168.12.223:80	1 sec		0 bps	3.22 KB
	HTTP	TCP	192.168.12.234:58380	192.168.12.223:80	1 sec		0 bps	3.22 KB
	HTTP	TCP	192.168.16.189:58910	192.168.12.223:80	1 sec		0 bps	3.22 KB
	HTTP	TCP	192.168.11.218:58369	192.168.12.223:80	2 sec		0 bps	2.3 KB
	HTTP	TCP	192.168.16.229:58673	192.168.12.223:80	2 sec		0 bps	2.3 KB

图 2-7 Ntopng 对活跃数据流的统计

通过图 2-7 可以很清晰地看出每条数据流的发送方和接收方，而“Breakdown”列展示了发送和接收数据量的大小，单击右上角的“Applications”按钮，还可以根据不同的应用类型，例如 HTTP、ICMP、DNS 等，有选择地查看活跃的数据流状态。

“Hosts”栏目显示了所有 Ntopng 可见的主机信息，可分类显示本地的或远程的主机列表，还可以显示每台主机间的交互信息、本地主机矩阵图等信息，如图 2-8 所示。

All Hosts						
IP Address	Location	Name	Seen Since	Breakdown	Throughput▼	Traffic
192.168.12.223	Local	192.168.12.223	23 h, 46 min, 42 sec	Recv	5.56 Mbit	58.94 GB
192.168.18.77	Remote	host0077.bestjob.com	23 h, 46 min, 37 sec	Sent	465.29 Kbit	578.04 MB
192.168.18.80	Remote	host0080.bestjob.com	23 h, 46 min, 37 sec	Sent	363.17 Kbit	572.41 MB
192.168.18.32	Remote	host0032.bestjob.com	23 h, 46 min, 33 sec	Sent	363.08 Kbit	548.89 MB
192.168.18.98	Remote	host0098.bestjob.com	23 h, 46 min, 42 sec	Sent	345.72 Kbit	576.67 MB
192.168.18.93	Remote	host0093.bestjob.com	23 h, 46 min, 40 sec	Sent	345.7 Kbit	575.76 MB
192.168.18.78	Remote	host0078.bestjob.com	23 h, 46 min, 42 sec	Sent	345.7 Kbit	579.44 MB
192.168.18.63	Remote	host0063.bestjob.com	23 h, 46 min, 39 sec	Sent	290.38 Kbit	589.64 MB
192.168.18.86	Remote	host0086.bestjob.com	23 h, 46 min, 42 sec	Sent	287.58 Kbit	575.34 MB
192.168.18.100	Remote	host0100.bestjob.com	23 h, 46 min, 42 sec	Sent	287.57 Kbit	578.73 MB

图 2-8 Ntopng 收集到的所有主机列表

在图 2-8 中，Ntopng 展示了每台主机的主机名、IP 地址、主机所处地域（本地或者远程）、数据收集持续时长、发送 / 接收数据量、主机网络吞吐量、数据传输量等信息。如果想要了解每台主机更详细的统计信息，可以在图 2-8 中单击每台主机的 IP 进入主机详细信息页，如图 2-9 所示。

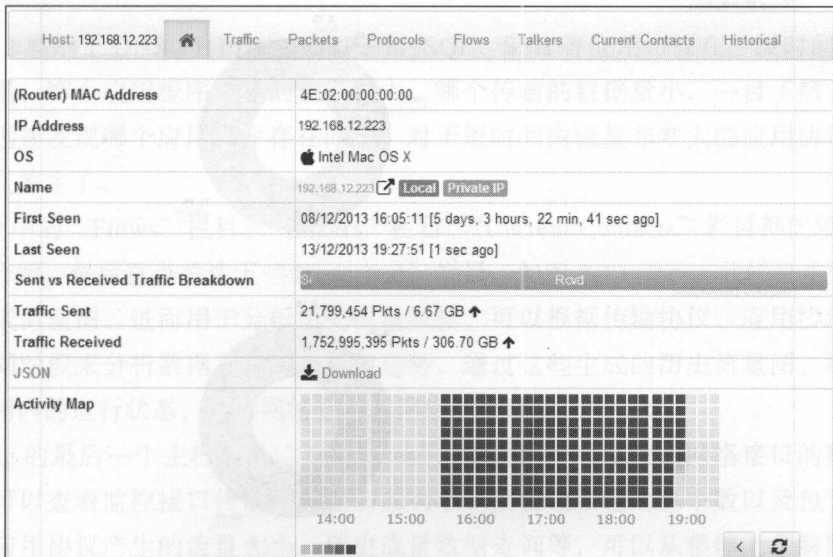


图 2-9 Ntopng 对每台主机的详细监控页面

从图 2-9 可以看出，每台主机的详细信息页中又分很多小栏目，默认打开的页面展示了此主机的 MAC 地址、IP 地址、操作系统类型、主机名、数据收集的开始和截止时间、数据发送和接收量等信息，单击图 2-9 中的“Traffic”栏目，可以根据协议类型查看数据的通信量，并且还通过饼状图进行了汇总，如图 2-10 所示。

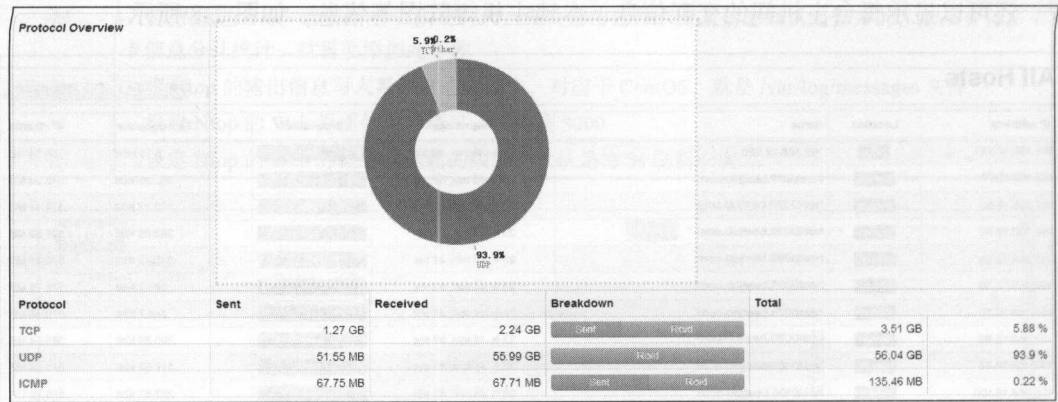


图 2-10 Ntopng 根据协议类型进行数据流量统计

在图 2-10 中，Ntopng 将通信流量以 TCP、UDP、ICMP 三种协议类型进行分别统计，并且通过饼状图方式进行整体汇总，这对于了解网络中某个通信协议的流量是非常有用的。

单击图 2-9 中的“Packets”栏目，可以根据发送、接收包的数量进行流量统计，如图 2-11 所示。

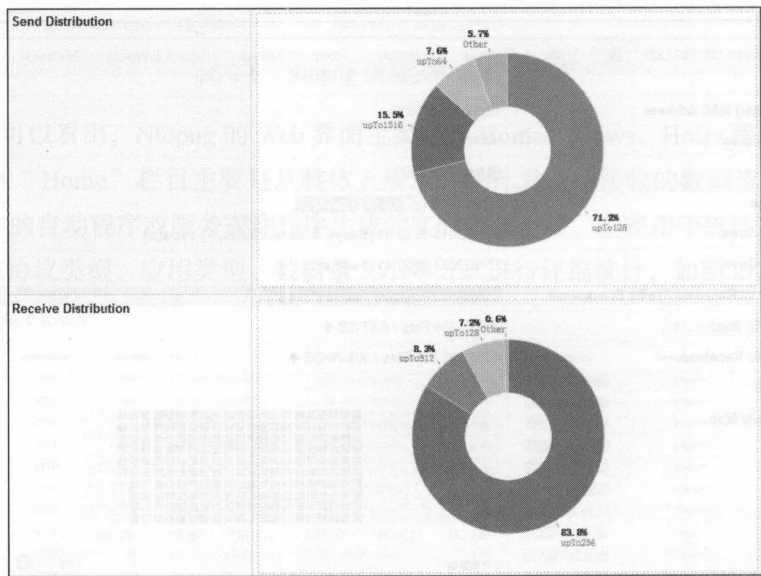


图 2-11 Ntopng 绘制的数据包发送量、接收量分布图

“Packets” 栏目展示的是数据包发送量、接收量的分布图。从图 2-11 中可以看到，在发送的数据包中，数据包量高达 128 的占总发送量的 71.2%，在接收的数据包中，数据包量高达 256 的占总接收量的 83.8%，这个功能可以帮助网络管理员判断网络中发送或接收数据包的数据及占据的比例，网络管理员可以据此来判断网络是否存在异常，进而解决潜在的网络问题。

单击图 2-9 中的 “Protocols” 栏目，可以根据应用程序的类型进行流量统计，如图 2-12 所示。

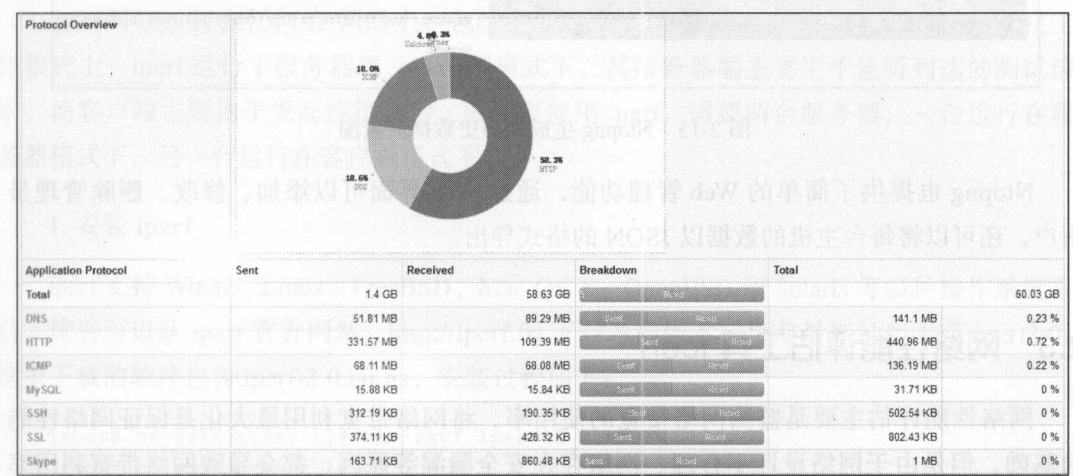


图 2-12 Ntopng 根据应用程序的类型进行流量统计

图 2-12 展示了 DNS、HTTP、ICMP、MySQL、SSH 等应用协议在一段时间内的发送、接收数据量，哪个应用程序传输的数据量大，哪个传输的数据量小，一目了然。通过此功能，可以迅速发现哪个应用程序存在问题，对于短时间内流量非常大的应用协议，管理员就需要重点关注了。

图 2-9 中的 “Flows” 栏目、“Talkers” 栏目、“Current Contacts” 栏目都比较浅显易懂，这里不再介绍。最后重点关注下 “Historical” 栏目，如图 2-13 所示，此栏目主要用于将数据流量生成流量图，进而用于分析历史流量数据，可以根据传输协议、应用协议类型等方式选择不同时段来分析数据发送、接收的趋势，通过这些生成的历史流量图，可以分析网络一段时间内的运行状态，并为网络管理和网络故障排除提供依据。

Ntopng 的最后一个主栏目是 “Interfaces”，主要用于对监控的网络接口的数据流量进行分析，可以查看监控接口传输数据量的总大小、接收数据包的总个数以及包大小分布状况、每个应用协议产生的流量大小、历史流量数据查询等，可以从整体上了解网络接口的通信状态。

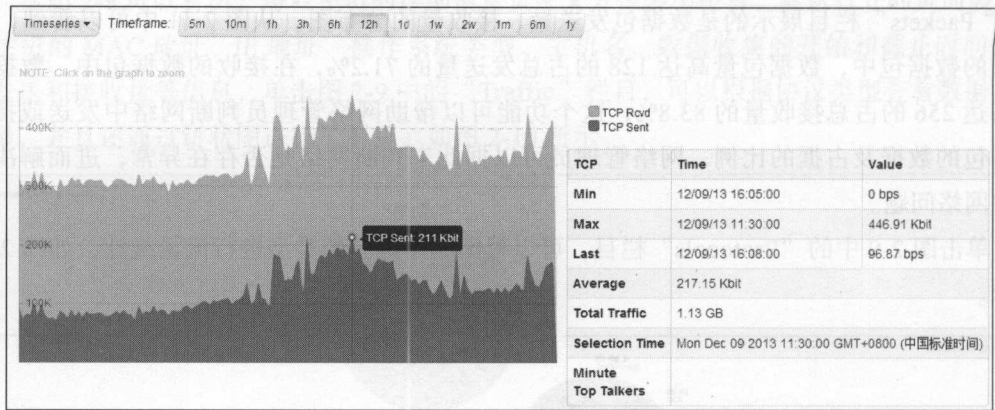


图 2-13 Ntopng 生成的历史数据流量图

Ntopng 也提供了简单的 Web 管理功能，通过 Web 界面可以添加、修改、删除管理员用户，还可以将每台主机的数据以 JSON 的格式导出。

2.3 网络性能评估工具 iperf

网络性能评估主要是监测网络带宽的使用率，将网络带宽利用最大化是保证网络性能的基础，但是由于网络设计不合理、网络存在安全漏洞等原因，都会导致网络带宽利用率低。要找到网络带宽利用率低的原因，可以对网络传输进行监控，此时就需要用到一些网络性能评估工具，而 iperf 就是这样一个网络带宽测试工具，本节将详细介绍 iperf 的使用方式。

2.3.1 iperf 能做什么

iperf 是一个基于 TCP/IP 和 UDP/IP 的网络性能测试工具，它可以用来测量网络带宽和网络质量，还可以提供网络延迟抖动、数据包丢失率、最大传输单元等统计信息。网络管理员可以根据这些信息了解并判断网络性能问题，从而定位网络瓶颈，解决网络故障。

下面介绍 iperf 的主要功能。

(1) TCP 方面

- ❑ 测试网络带宽。
- ❑ 支持多线程，在客户端与服务器端支持多重连接。
- ❑ 报告 MSS/MTU 值的大小。
- ❑ 支持 TCP 窗口值自定义并可通过套接字缓冲。

(2) UDP 方面

- 可以设置指定带宽的 UDP 数据流。
- 可以测试网络抖动值、丢包数。
- 支持多播测试。
- 支持多线程，在客户端与服务器端支持多重连接。

2.3.2 iperf 的安装与使用

iperf 可以运行在任何 IP 网络上，包括本地以太网、接入因特网、Wi-Fi 网络等。在工作模式上，iperf 运行于服务器端、客户端模式下，其服务器端主要用于监听到达的测试请求，而客户端主要用于发起连接会话，因此要使用 iperf，需要两台服务器，一台运行在服务器模式下，另一台运行在客户端模式下。

1. 安装 iperf

iperf 支持 Win32、Linux、FreeBSD、Mac OS X、OpenBSD 和 Solaris 等多种操作系统平台。读者可以从 iperf 官方网站：<http://iperf.fr/> 下载各种版本，目前最新的版本是 iperf3.0，这里下载的软件包为 iperf-3.0.tar.gz，安装过程如下：

```
[root@ networkserver ~]# tar zxvf iperf-3.0.tar.gz
[root@ networkserver ~]# cd iperf
[root@ networkserver ~]# ./configure
[root@ networkserver iperf]# make
[root@ networkserver iperf]# make install
```

这样，iperf 就安装完成了。

2. iperf 参数介绍

在完成 iperf 安装后，执行“iperf3 -h”即可显示 iperf 的详细用法。iperf 的命令行选项共分为三类，分别是客户端与服务器端公用选项、服务器端专用选项和客户端专用选项，下面对常用的选项进行介绍。

服务器端专用选项的含义如表 2-4 所示。

表 2-4 服务器端专用选项的含义

命令行参数	含义描述
-s	将 iperf 以服务器端模式启动，例如：iperf3 -s，iperf3 默认启动的监听端口为 5201，可以通过“-p”选项修改默认监听端口
-D	将 iperf 作为后台守护进程运行，例如：iperf3 -s -D

客户端专用选项的含义如表 2-5 所示。

表 2-5 客户端专用选项的含义

命令行参数	含义描述
-c	将 iperf 以客户端模式启动 例如：iperf3 -c 192.168.12.168，其中 192.168.12.168 是服务器端的 IP 地址
-u	指定使用 UDP 协议
-b [K M G]	指定 UDP 模式使用的带宽，单位 bits/sec。此选项与“-u”选项相关。默认值是 1 Mbits/sec
-t	指定传输数据包的总时间。iperf 将在指定的时间内，重复发送指定长度的数据包。默认是 10s
-n [K M G]	指定传输数据包的字节数，例如：iperf3 -c 192.168.12.168 -n 100M
-l	指定读写缓冲区的长度。TCP 方式默认大小为 8KB，UDP 方式默认大小为 1470B
-P	指定客户端与服务器端之间使用的线程数。默认是 1 个线程。需要客户端与服务器端同时使用此参数
-R	切换数据发送、接收模式，例如默认客户端发送，服务器端接收，设置此参数后，数据流向变为客户端接收，服务器端发送
-w	指定套接字缓冲区大小，在 TCP 方式下，此设置为 TCP 窗口的大小。在 UDP 方式下，此设置为接受 UDP 数据包的缓冲区大小，用来限制可以接收数据包的最大值
-B	用来绑定一个主机地址或接口，这个参数仅用于具有多个网络接口的主机。在 UDP 模式下，此参数用于绑定和加入一个多播组
-M	设置 TCP 最大信息段的值
-N	设置 TCP 无延时

客户端与服务器端公用选项的含义如表 2-6 所示。

表 2-6 客户端与服务器端公用选项的含义

命令行参数	含义描述
-f [k m g K M G]	指定带宽输出单位，“[k m g K M G]”分别表示以 Kbits、Mbits、Gbits、KBytes、MBytes、GBytes 显示输出结果，默认以 Mbits 为单位，例如：iperf3 -c 192.168.12.168 -f M
-p	指定服务器端使用的端口或客户端所连接的端口，例如： iperf3 -s -p 9527 iperf3 -c 192.168.12.168 -p 9527
-i	指定每次报告之间的时间间隔，单位为秒。如果设置为非零值，就会按照此时间间隔输出测试报告。默认值为 1 例如：iperf3 -c 192.168.12.168 -i 2
-F	指定文件作为数据流进行带宽测试 例如：iperf3 -c 192.168.12.168 -F web-ixdba.tar.gz

2.3.3 iperf 应用实例

要使用 iperf，首先启动一个服务器端，这里假定服务器端的 IP 地址为 192.168.12.168，在此服务器上运行“iperf3 -s”即可开启 iperf 的服务器端模式。在默认情况下，iperf3 将在服务器端打开一个 5201 监听端口，此时就可以将另一台服务器作为客户端执行 iperf 功能测试。

1. 测试 TCP 吞吐量

为了确定网卡的最大吞吐量，可以在任意客户端运行 `iperf` 命令，`iperf` 将尝试从客户端尽可能快地向服务器端发送数据请求，并且会输出发送的数据量和网卡平均带宽值。图 2-14 展示了通过一个最简单的带宽测试命令的输出结果。

```
[root@networkserver app]# iperf3 -c 192.168.12.168
Connecting to host 192.168.12.168, port 5201
[ 4] local 192.168.12.123 port 51628 connected to 192.168.12.168 port 5201
[ ID] Interval           Transfer     Bandwidth       Retransmits
[ 4] 0.00-1.00 sec      111 MBytes  929 Mbits/sec    3
[ 4] 1.00-2.00 sec      113 MBytes  944 Mbits/sec    0
[ 4] 2.00-3.00 sec      112 MBytes  944 Mbits/sec    0
[ 4] 3.00-4.00 sec      112 MBytes  941 Mbits/sec    0
[ 4] 4.00-5.00 sec      112 MBytes  944 Mbits/sec    0
[ 4] 5.00-6.00 sec      112 MBytes  943 Mbits/sec    0
[ 4] 6.00-7.00 sec      112 MBytes  943 Mbits/sec    0
[ 4] 7.00-8.00 sec      112 MBytes  944 Mbits/sec    8
[ 4] 8.00-9.00 sec      112 MBytes  942 Mbits/sec    0
[ 4] 9.00-10.00 sec     112 MBytes  937 Mbits/sec    0

--
[ ID] Interval           Transfer     Bandwidth       Retransmits
[ 4] 0.00-10.00 sec     1.10 GBytes  941 Mbits/sec   11
[ 4] 0.00-10.00 sec     1.09 GBytes  940 Mbits/sec
sender
receiver
iperf Done.
```

图 2-14 通过 `iperf` 测试网络带宽利用率

从图 2-14 可以看出，`iperf` 默认的运行时间是 10s，每隔 1s 输出一次传输状态，同时还可以看到每秒传输的数据量在 112MB 左右，刚好与“Bandwidth”列的值对应起来，网卡的带宽速率维持在 941Mbits/sec 左右，而测试的服务器是千兆网卡，这个测试值也基本合理。在输出的最后，`iperf` 还给出了总的发送、接收量，以及带宽速率平均值，通过这些值，基本可以判断网络带宽是否正常，网络传输状态是否稳定。

`iperf` 提供很多参数，可以多角度、全方位地测试网络带宽利用率，例如，要改变 `iperf` 运行的时间和输出频率，可以通过“-t”和“-i”参数来实现，如图 2-15 所示。

```
[root@networkserver app]# iperf3 -c 192.168.12.168 -t 20 -i 5
Connecting to host 192.168.12.168, port 5201
[ 4] local 192.168.12.123 port 55771 connected to 192.168.12.168 port 5201
[ ID] Interval           Transfer     Bandwidth       Retransmits
[ 4] 0.00-5.00 sec      560 MBytes  940 Mbits/sec   31
[ 4] 5.00-10.00 sec     562 MBytes  942 Mbits/sec   18
[ 4] 10.00-15.00 sec     561 MBytes  941 Mbits/sec    5
[ 4] 15.00-20.00 sec     561 MBytes  942 Mbits/sec   19

--
[ ID] Interval           Transfer     Bandwidth       Retransmits
[ 4] 0.00-20.00 sec     2.19 GBytes  941 Mbits/sec   73
[ 4] 0.00-20.00 sec     2.19 GBytes  941 Mbits/sec
sender
receiver
iperf Done.
```

图 2-15 添加“-t”和“-i”参数后的 `iperf` 输出

从图 2-15 可以看出, 输出状态的间隔变为每 5s 一次, 总共执行测试时间为 20s, 测试的带宽速率仍然保持在 941Mbps/sec 左右, 唯一变化的是失败重传次数增加了。

为了模拟大量的数据传输, 也可以指定要发送的数据量, 这可以通过“-n”参数来实现。在指定“-n”参数后,“-t”参数失效, iperf 在传输完毕指定大小的数据包后, 自动结束, 如图 2-16 所示。

```
[root@networkserver app]# iperf3 -c 192.168.12.168 -i 10 -n 5000000000
Connecting to host 192.168.12.168, port 5201
[ 4] local 192.168.12.123 port 58611 connected to 192.168.12.168 port 5201
[ ID] Interval            Transfer          Bandwidth        Retransmits
[ 4] 0.00-10.00 sec      1.09 GBytes      941 Mbits/sec    103
[ 4] 10.00-20.00 sec     1.10 GBytes      942 Mbits/sec     45
[ 4] 20.00-30.01 sec     1.10 GBytes      941 Mbits/sec     39
[ 4] 30.01-40.00 sec     1.06 GBytes      907 Mbits/sec    484
[ 4] 40.00-42.86 sec      321 MBytes       942 Mbits/sec     38
- - - - -
[ ID] Interval            Transfer          Bandwidth        Retransmits
[ 4] 0.00-42.86 sec      4.66 GBytes       933 Mbits/sec    709
[ 4] 0.00-42.86 sec      4.66 GBytes       933 Mbits/sec
sender
receiver
iperf Done.
```

图 2-16 iperf 客户端通过“-n”参数指定要传输的数据量

图 2-16 的例子是指定发送一个 5GB 左右的数据包, 并且每隔 10s 输出一次传输状态, 从这个输出可以看出, 当失败重传次数较多时, 传输速率急速下降。

有时候, 为了模拟更真实的 TCP 应用, iperf 客户端允许从一个特定的文件发送数据, 这可以通过“-F”参数实现, 如图 2-17 所示。

```
[root@networkserver app]# iperf3 -c 192.168.12.168 -F webdata.tar.gz -i 5 -t 20
Connecting to host 192.168.12.168, port 5201
[ 4] local 192.168.12.123 port 45894 connected to 192.168.12.168 port 5201
[ ID] Interval            Transfer          Bandwidth        Retransmits
[ 4] 0.00-5.00 sec        560 MBytes       940 Mbits/sec     2
[ 4] 5.00-10.00 sec       561 MBytes       942 Mbits/sec     5
[ 4] 10.00-13.48 sec      390 MBytes       941 Mbits/sec     5
- - - - -
[ ID] Interval            Transfer          Bandwidth        Retransmits
[ 4] 0.00-13.48 sec      1.48 GBytes       941 Mbits/sec    12
Sent 1.48 GByte / 1.48 GByte (100%) of webdata.tar.gz
[ 4] 0.00-13.48 sec      1.48 GBytes       941 Mbits/sec
sender
receiver
iperf Done.
```

图 2-17 iperf 客户端通过“-F”参数指定文件来发送数据

在图 2-17 的例子中, 通过“-F”参数指定了一个 webdata.tar.gz 文件作为 iperf 要传输的数据。在使用此参数时, 需要同时指定一个“-t”参数来设置要测试传输的时间, 这个时间尽量设置长一些, 因为在默认传输时间 10s 内, 这个文件可能还没有传完。

在使用 iperf 进行网络带宽测试时, 如果没有指定发送方式, iperf 客户端只会使用单一的线程, 而 iperf 是支持多线程的, 可以使用 iperf 提供的“-P”参数来设置多线程的数目,

通过使用多线程，可以在一定程度上增加网络的吞吐量。

下面通过两个例子进行简单对比，图 2-18 是 iperf 使用单线程传输 1.86GB 数据所消耗的时间和带宽使用情况。为了速率单位统一，这里使用“-f”参数将输出结果都通过 MBytes/sec 来显示。

```
[root@networkserver app]# iperf3 -c 192.168.12.168 -n 2000000000 -i 5 -f M
Connecting to host 192.168.12.168, port 5201
[ 4] local 192.168.12.123 port 48833 connected to 192.168.12.168 port 5201
[ ID] Interval            Transfer        Bandwidth      Retransmits
[ 4] 0.00-5.00 sec        560 MBytes     112 MBytes/sec 3
[ 4] 5.00-10.01 sec       562 MBytes     112 MBytes/sec 6
[ 4] 10.01-15.01 sec      561 MBytes     112 MBytes/sec 7
[ 4] 15.01-17.00 sec      224 MBytes     112 MBytes/sec 4
-----
[ ID] Interval            Transfer        Bandwidth      Retransmits
[ 4] 0.00-17.00 sec      1.86 GBytes     112 MBytes/sec 20
[ 4] 0.00-17.00 sec      1.86 GBytes     112 MBytes/sec
sender
receiver
iperf Done.
```

图 2-18 iperf 在单线程模式下的传输时间和传输速率

从图 2-18 中可以看出，传输 1.86GB 的数据消耗了 17s 的时间，平均带宽速率为 112MBytes/sec（注意单位）。下面再看看使用多线程后，iperf 传输同样大小数据量所消耗的时间和平均带宽速率，如图 2-19 所示。

```
[root@networkserver app]# iperf3 -c 192.168.12.168 -n 2000000000 -i 5 -P 2 -f M
Connecting to host 192.168.12.168, port 5201
[ 4] local 192.168.12.123 port 48909 connected to 192.168.12.168 port 5201
[ 6] local 192.168.12.123 port 48910 connected to 192.168.12.168 port 5201
[ ID] Interval            Transfer        Bandwidth      Retransmits
[ 4] 0.00-5.00 sec        430 MBytes     86.0 MBytes/sec 1618
[ 6] 0.00-5.00 sec        430 MBytes     86.0 MBytes/sec 1756
[SUM] 0.00-5.00 sec      861 MBytes     172 MBytes/sec 3374
-----
[ 4] 5.00-10.00 sec      446 MBytes     89.2 MBytes/sec 1925
[ 6] 5.00-10.00 sec      446 MBytes     89.2 MBytes/sec 1856
[SUM] 5.00-10.00 sec    892 MBytes     178 MBytes/sec 3781
-----
[ 4] 10.00-10.79 sec     77.2 MBytes     98.3 MBytes/sec 582
[ 6] 10.00-10.79 sec     77.2 MBytes     98.3 MBytes/sec 556
[SUM] 10.00-10.79 sec  154 MBytes     197 MBytes/sec 1138
-----
[ ID] Interval            Transfer        Bandwidth      Retransmits
[ 4] 0.00-10.79 sec      954 MBytes     88.4 MBytes/sec 4125
[ 4] 0.00-10.79 sec      954 MBytes     88.4 MBytes/sec
receiver
[ 6] 0.00-10.79 sec      954 MBytes     88.4 MBytes/sec 4168
[ 6] 0.00-10.79 sec      954 MBytes     88.4 MBytes/sec
receiver
[SUM] 0.00-10.79 sec  1.86 GBytes     177 MBytes/sec 8293
[SUM] 0.00-10.79 sec  1.86 GBytes     177 MBytes/sec
sender
receiver
iperf Done.
```

图 2-19 iperf 使用多线程后的数据传输状态

这里通过“-P”参数开启了 2 个多线程，从传输时间上看，传输 1.86GB 的数据，消耗时间为 10.79s，比之前单线程的传输时间少了近 7s，在平均带宽速率上，从之前单线程的

112MBytes/sec 提高到 177MBytes/sec，从这个结果可以看出，多线程对网络传输性能的提高不小。

2. 测试 UDP 丢包和延迟

iperf 也可以用于 UDP 数据包吞吐量、丢包率和延迟指标，但是由于 UDP 协议是一个非面向连接的轻量级传输协议，并且不提供可靠的数据传输服务，因此对 UDP 应用的关注点不是传输数据有多快，而是它的丢包率和延时指标。通过 iperf 的 “-u” 参数即可测试 UDP 应用的传输性能，图 2-20 测试的是在 iperf 客户端传输 100MB 的 UDP 数据包的输出结果。

```
[root@networkserver app]# iperf3 -c 192.168.12.168 -u -b 100M -f M -i 3
Connecting to host 192.168.12.168, port 5201
[ 4] local 192.168.12.123 port 45516 connected to 192.168.12.168 port 5201
[ ID] Interval      Transfer    Bandwidth  Total Datagrams
[ 4]  0.00-3.00    sec  36.3 MBytes 12.1 MBytes/sec 4650
[ 4]  3.00-6.00    sec  37.5 MBytes 12.5 MBytes/sec 4800
[ 4]  6.00-9.00    sec  37.5 MBytes 12.5 MBytes/sec 4800
[ 4]  9.00-10.00   sec  12.5 MBytes 12.5 MBytes/sec 1600
-----
[ ID] Interval      Transfer    Bandwidth  Jitter    Lost/Total Datagrams
[ 4]  0.00-10.00   sec  124 MBytes 12.4 MBytes/sec 0.052 ms  53/15850 (0.33%)
[ 4] Sent 15850 datagrams
iperf Done.
```

图 2-20 iperf 传输 100MB 的 UDP 数据包的输出结果

在图 2-20 中，重点关注虚线下的一段内容，在这段输出中，“Jitter” 列表示抖动时间，或者称为传输延迟，“Lost/Total” 列表示丢失的数据包和总的数据包数量，后面的 0.33% 是平均丢包的比率，“Datagrams” 列显示的是总共传输数据包的数量。

这个输出结果过于简单，要了解更详细的 UDP 丢包和延时信息，可以在 iperf 服务端查看，因为在客户端执行传输测试的同时，服务器端也会同时显示传输状态，如图 2-21 所示。

```
[root@server app]# iperf3 -s -i 3
Server listening on 5201
-----
Accepted connection from 192.168.12.123, port 45108
[ 5] local 192.168.12.168 port 5201 connected to 192.168.12.123 port 45516
[ ID] Interval      Transfer    Bandwidth  Jitter    Lost/Total Datagrams
[ 5]  0.00-3.00    sec  36.2 MBytes 101 Mbits/sec 0.053 ms  21/4650 (0.45%)
[ 5]  3.00-6.00    sec  37.5 MBytes 105 Mbits/sec 0.047 ms  0/4800 (0%)
[ 5]  6.00-9.00    sec  37.4 MBytes 105 Mbits/sec 0.047 ms  9/4800 (0.19%)
[ 5]  9.00-10.04   sec  12.3 MBytes 99.4 Mbits/sec 0.052 ms  23/1600 (1.4%)
-----
[ ID] Interval      Transfer    Bandwidth  Jitter    Lost/Total Datagrams
[ 5]  0.00-10.04   sec  124 MBytes 103 Mbits/sec 0.052 ms  53/15850 (0.33%)
```

图 2-21 iperf 服务器端显示的 UDP 传输状态

在这个输出中，详细记录了在传输过程中，每个阶段的传输延时和丢包率，在 UDP 应用中随着传输数据的增大，丢包率和延时也随之增加。对于延时和丢包可以通过改变应用

程序来缓解或修复,例如视频流应用,通过缓存数据的方式可以容忍更大的延时。

2.4 网络探测和安全审核工具 nmap

对于 nmap,相信很多安全运维人员并不陌生,它曾经在电影《黑客帝国》中出现过,是黑客和网络安全人员经常用到的工具,本节重点介绍此工具的实现原理和使用技巧。

2.4.1 nmap 和 Zenmap 简介

nmap 是一个开源免费的网络发现工具,通过它能够找出网络上在线的主机,并测试主机上哪些端口处于监听状态,接着通过端口确定主机上运行的应用程序类型与版本信息,最后利用它还能侦测出操作系统的类型和版本。由此可见, nmap 是一个功能非常强大的网络探测工具,同时它也成为网络黑客的最爱,因为 nmap 所实现的这些功能正是黑客入侵网络的一个基本过程。站在安全运维的角度,只有了解了黑客入侵的基本方式和过程,才能有目的、有针对性地进行安全防护,这也正是本节重点介绍 nmap 这个网络嗅探工具的原因。

nmap 是 Network Mapper 的缩写,由 Fyodor 在 1997 年创建,现在已经成为网络安全必备的工具之一,目前最新的版本为 nmap6.40,更多详细信息可以参考官方网站: www.nmap.org。nmap 作为一个流行的安全工具,它的主要特点有:

- ❑ 非常灵活。nmap 支持 10 多种扫描方式,并支持多种目标对象扫描。
- ❑ 支持主流操作系统。nmap 支持 Windows、Linux、BSD、Solaris、AIX、Mac OS 等多种平台,可移植性强。
- ❑ 使用简单。nmap 安装、使用都非常简单,基本用法就能满足一般使用需求。
- ❑ 自由软件。nmap 是在 GPL 协议下发布的,在 GPL License 的范围内可自由使用。

Zenmap 是 nmap 的 GUI 版本,由 nmap 官方提供,通常随着 nmap 安装包一起发布。Zenmap 是用 Python 语言编写的,能够在 Windows、Linux、UNIX、Mac OS 等不同系统上运行。开发 Zenmap 的目的主要是为 nmap 提供更加简单的操作方式。

2.4.2 nmap 基本功能与结构

nmap 功能非常强大,从它实现功能的方向性来划分,主要有如下四个基本功能:

- ❑ 主机发现。
- ❑ 端口扫描。
- ❑ 应用程序及版本侦测。
- ❑ 操作系统及版本侦测。

这四个基本功能既相互独立，又依次依赖，因为一般的网络嗅探都是从主机发现开始的，在发现在线的主机后，接着需要进行端口扫描，进而通过扫描到的端口确定运行的应用程序类型及版本信息，并最终确定操作系统的版本及漏洞信息。另外 nmap 还提供了防火墙与入侵检测系统的规避技巧，这个功能可以应用到基本功能的各个阶段中。最后，nmap 还提供了高级用法，即通过 NSE (Nmap Scripting Language) 脚本引擎功能对 nmap 基本功能进行补充和扩展。

2.4.3 nmap 的安装与验证

nmap 的安装非常简单，官方提供源码编译安装和 RPM 包两种方式，读者可根据自己的喜好选择安装即可。这里下载的版本为 nmap-6.40.tar.bz2，下面分别介绍两种安装方式。

1. 源码编译安装

从官方网站下载源码包，然后编译安装即可。编译安装过程无需额外参数，操作如下：

```
[root@localhost ~]# tar jxvf nmap-6.40.tar.bz2
[root@localhost ~]# cd nmap-6.40
[root@localhost nmap-6.40]# ./configure
[root@localhost nmap-6.40]# make
[root@localhost nmap-6.40]# make install
```

至此，源码编译编译方式安装 nmap 完成。

2. RPM 包安装

nmap 官方也提供 RPM 格式的安装包，直接从网站下载 RPM 格式的安装包，然后进行安装即可，操作过程如下：

```
[root@localhost ~]# wget http://nmap.org/dist/nmap-6.40-1.x86_64.rpm
[root@localhost ~]# rpm -Uvh nmap-6.40-1.x86_64.rpm
```

在完成安装后，执行“nmap -h”，如果能输出帮助信息，表示安装成功，否则根据错误提示重新安装。

2.4.4 nmap 的典型用法

前面提到了 nmap 主要包含四个方面的扫描功能，在详细介绍每个功能之前，首先介绍 nmap 的典型用法。最简单的 nmap 命令形式如下：

nmap 目标主机

通过这个命令，可以确定目标主机的在线情况和端口的监听状态，如图 2-22 所示。

```

[root@localhost ~]# nmap 192.168.12.189
Starting Nmap 6.40 ( http://nmap.org ) at 2013-12-20 15:01 CST
Nmap scan report for bogon (192.168.12.189)
Host is up (0.000089s latency).
Not shown: 997 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
21/tcp    open  ftp
3306/tcp   open  mysql
MAC Address: 90:B1:1C:41:28:E4 (Dell)
Nmap done: 1 IP address (1 host up) scanned in 0.21 seconds

```

图 2-22 nmap 的典型用法

由输出可知，目标主机“192.168.12.189”处于“up”状态，并且此主机上开放了22、21、3306端口，同时还侦测到每个端口对应的服务，在最后还给出目标主机网卡的MAC信息。

如果希望了解目标主机更多的信息，可以通过完全扫描的方式实现，nmap命令内置了“-A”选项，可以实现对目标主机进行主机发现、端口扫描、应用程序与版本侦测、操作系统识别等完整全面的扫描，命令形式如下：

```
nmap -T4 -A -v 目标主机
```

其中，“-A”选项用于开启全面扫描；“-T4”指定扫描过程中使用的时序模板，总共有6个等级（0~5），等级越高，扫描速度越快，但也越容易被防火墙或者入侵检测设备发现并屏蔽，所以选择一个适当的扫描等级非常重要，这里推荐使用“-T4”；“-v”参数可显示扫描细节。图2-23是nmap对某主机的全面扫描过程。

```

[root@localhost ~]# nmap -A -T4 192.168.12.188
Starting Nmap 6.40 ( http://nmap.org ) at 2013-12-20 16:26 CST
Nmap scan report for bogon (192.168.12.188)
Host is up (0.000096s latency).
Not shown: 997 closed ports
PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 4.3 (protocol 2.0)
| ssh-hostkey: 1024 a0:b7:70:c7:ec:42:31:dc:be:37:c9:32:e2:b5:2b:82 (DSA)
|_ 2048 a3:2a:59:8f:82:9f:d9:c5:5b:08:00:5d:96:34:b3 (RSA)
111/tcp   open  rpcbind  2 (RPC #100000)
| rpcinfo:
|_  program version  port/proto  service
|_  100000  2          111/tcp     rpcbind
|_  100000  2          111/udp     rpcbind
|_  100024  1          831/udp     status
|_  100024  1          834/tcp     status
80/tcp    open  http     Apache/2.0.63 ((Unix) DAV/2 mod_jk/1.2.14 PHP/5.2.5)
|_ http-methods: Potentially risky methods: TRACE
|_ See http://nmap.org/nmapdoc/scripts/http-methods.html
|_ http-robots.txt: 5 disallowed entries
|_ /logs/ /conf/ /css/ /WEB-INF/ /js/
|_ http-title: Site doesn't have a title (text/html).
MAC Address: 90:B1:1C:41:28:E6 (Dell)
Device type: general purpose
Running: Linux 2.6.X
OS CPE: cpe:/o:linux:linux_kernel:2.6
OS details: Linux 2.6.18 - 2.6.32
Network Distance: 1 hop

TRACEROUTE
HOP RTT ADDRESS
1 0.10 ms bogon (192.168.12.188)

```

图 2-23 nmap 对主机“192.168.12.188”的全面扫描过程

从图 2-23 中可以看出，整个扫描过程非常详细：第一部分是对主机是否在线进行扫描；第二部分是对端口进行扫描，在默认情况下 nmap 会扫描 1000 个最有可能开放的端口，由于只扫描到 22、111、80 三个端口处于打开状态，所以在输出中会有“997 closed ports”的描述；第三部分是对端口上运行的应用服务以及版本号进行统计，可以看到，扫描结果非常详细地记录了软件的版本信息；第四部分是对操作系统类型和版本进行探测，从扫描结果来看，还是非常准确的；第五部分是对目标主机的路由跟踪信息。

2.4.5 nmap 主机发现扫描

主机发现主要用来判断目标主机是否在线，其扫描原理类似于 ping 命令，通过发送探测数据包到目标主机，如果能收到回复，那么认为目标主机处于在线状态。nmap 支持多种不同的主机探测方法，例如发送 TCP SYN/ACK 包、发送 SCTP 包、发送 ICMP echo/timestamp/netmask 请求报文等，用户可在不同的环境下选择不同的方式来探测目标主机。

1. 主机发现的用法

nmap 提供了丰富的选项以供用户选择不同的主机发现探测方式，使用语法如下：

```
nmap [ 选项或参数 ] 目标主机
```

nmap 常用的主机发现选项与含义如表 2-7 所示。

表 2-7 常用的主机发现选项及含义

选项	含义
-sn	只进行主机发现扫描，不进行端口扫描
-Pn	跳过主机发现扫描，将所有指定的主机都视为在线状态，进行端口扫描
-sL	仅仅列出指定的目标主机 IP，不进行主机发现扫描
-PS/PA/PU/PY[portlist]	指定 nmap 使用 TCP SYN、TCP ACK、UDP、SCTP 方式进行主机发现，例如 nmap -PS80,21
-PE/PP/PM	指定 nmap 使用 ICMP echo、timestamp、netmask 请求报文方式发现主机
-PO	使用 IP 协议包探测目标主机是否在线
-n/-R	指定是否使用 DNS 解析，其中，“-n”表示不进行 DNS 解析；“-R”表示总是进行 DNS 解析

在这些选项中，比较常用的是“-sn”和“-Pn”，例如，查看某个网段有哪些主机在线，就需要使用“-sn”选项，而如果已经知道了目标主机在线，仅仅想扫描主机开放的端口时，就需要用“-Pn”选项。

2. 使用实例

下面以探测 www.abc.com 主机的信息为例，简单演示主机发现的用法。首先，在联网

的服务器上执行如下命令：

```
nmap -sn -PE -PS22,80 -PU53 www.abc.com
```

执行结果如图 2-24 所示。

```
[root@localhost ~]# nmap -sn -PE -PS22,80 -PU53 www.abc.com
Starting Nmap 6.40 ( http://nmap.org ) at 2013-12-20 20:08 CST
Nmap scan report for www.abc.com (61.185.133.234)
Host is up (0.0072s latency).
Other addresses for www.abc.com (not scanned): 116.34.65.215
rDNS record for 61.185.133.234: 234.133.185.61.broad.ak.sn.dynamic.abc.com.cn
Nmap done: 1 IP address (1 host up) scanned in 6.52 seconds
```

图 2-24 主机发现的执行结果

在这个例子中，使用了“-PE”、“-PS”、“-PU”等参数，根据上面的介绍，“-PE”是以发送 ICMP echo 报文的形式进行主机探测的，“-PS”是以发送 TCP SYN/ACK 包的形式侦测主机信息的，而“-PU”则是以 UDP 的方式进行主机侦测的。为了清晰展示 nmap 的侦测方式和侦测过程，这里通过抓包工具 Wireshark 动态监测 nmap 探测主机的过程，如图 2-25 所示。

```
4.734372 192.168.12.188 -> 61.185.133.234 ICMP Echo (ping) request
4.734388 192.168.12.188 -> 61.185.133.234 TCP 36910 > ssh [SYN] Seq=0 Win=1024 Len=0 MSS=1460
4.734398 192.168.12.188 -> 61.185.133.234 TCP 36910 > http [SYN] Seq=0 Win=1024 Len=0 MSS=1460
4.734406 192.168.12.188 -> 61.185.133.234 DNS Server status request
4.741577 61.185.133.234 -> 192.168.12.188 TCP ssh > 36910 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
4.741730 61.185.133.234 -> 192.168.12.188 ICMP Echo (ping) reply
4.741829 61.185.133.234 -> 192.168.12.188 ICMP Destination unreachable (Port unreachable)
4.742009 61.185.133.234 -> 192.168.12.188 TCP http > 36910 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460
```

图 2-25 通过 Wireshark 获取的 nmap 探测主机过程

从图 2-25 中可以看到，nmap 所在的主机“192.168.12.188”向目标主机“61.185.133.234”发送了四个探测包，分别是 ICMP Echo (ping)、22 端口和 80 端口的 TCP SYN、53 端口的 UDP 包。但是仅仅收到了 ICMP Echo (ping) 和 80 端口的回复，22 端口返回了“RST”标识，这说明 22 端口处于关闭状态。不过 nmap 的原则是只要能收到任何一种探测请求的回复，就认为此主机处于在线状态。

2.4.6 nmap 端口扫描

端口扫描是 nmap 最核心的功能，通过端口扫描可以发现目标主机上 TCP、UDP 端口的开放情况。nmap 在默认状态下会扫描 1000 个最有可能开放的端口，并将侦测到的端口状态分为 6 类，分别是：

- ☐ open，表示端口是开放的。
- ☐ closed，表示端口是关闭的。
- ☐ filtered，表示端口被防火墙屏蔽，无法进一步确定状态。

❑ `unfiltered`，表示端口没有被屏蔽，但是否处于开放状态，还需要进一步确定。

❑ `open|filtered`，表示不确定状态，端口可能是开放的，也可能是屏蔽的。

❑ `closed|filtered`，表示不确定状态，端口可能是关闭的，也可能是屏蔽的。

在端口扫描方式上，`nmap` 支持 10 多种探测方法，最常用的有“TCP SYN scanning”，这是默认的端口扫描方式，另外还有“TCP connect scanning”、“TCP ACK scanning”、“TCP FIN/Xmasscanning”、“UDP scanning”等探测方式。具体使用哪种探测方式，用户可自己指定。

1. 端口扫描的用法

`nmap` 提供了多个选项以供用户来指定扫描方式和扫描端口，使用语法如下：

`nmap` [选项或参数] 目标主机

`nmap` 端口扫描的常用选项与含义如表 2-8 所示。

表 2-8 nmap 端口扫描的常用选项及含义

选项	含义
<code>-sS/sT/sA/sW/sM</code>	指定使用 TCP SYN/Connect()/ACK/Window/Maimon scans 来对目标主机进行端口扫描
<code>-sU</code>	指定使用 UDP 扫描方式扫描目标主机的 UDP 端口状况
<code>-sN/sF/sX</code>	指定使用 TCP Null、FIN、Xmas scans 秘密扫描方式来协助侦测目标主机的 TCP 端口状态
<code>-p <port ranges></code>	仅仅扫描指定的一个或一批端口 例如“ <code>-p80</code> ”、“ <code>-p1-100</code> ”、“ <code>-p T:80-88,8000,8080,U:53,111,S:9</code> ”，其中 T 表示 TCP 协议，U 表示 UDP 协议，S 表示 SCTP 协议
<code>-F</code>	快速扫描模式，仅仅扫描开放率最高的前 100 个端口
<code>--top-ports <number></code>	仅扫描开放率最高的 number 个端口

2. 使用实例

下面仍以探测 `www.abc.com` 主机的信息为例，简单演示端口扫描的使用方法。首先，在联网的服务器上执行如下命令：

`nmap -sU -sS -F www.abc.com`

执行结果如图 2-26 所示。

在图 2-26 中，参数“`-sS`”表示使用 TCP SYN 方式扫描 TCP 端口，“`-sU`”表示扫描 UDP 端口，“`-F`”表示使用快速扫描模式，扫描最可能开放的前 100 个端口（TCP 和 UDP 各 100 个端口），由输出可知，有 21 个端口处于开放或者屏蔽状态，其他 179 个端口处于关闭状态。

```

[root@localhost ~]# nmap -sU -sS -F www.abc.com
Nmap scan report for www.abc.com (117.185.133.232)
Host is up (0.0071s latency).
Not shown: 179 closed ports
PORT      STATE      SERVICE
21/tcp    filtered  ftp
80/tcp    open      http
88/tcp    open      kerberos-sec
111/tcp   filtered  rpcbind
135/tcp   filtered  msrpc
443/tcp   open      https
514/tcp   filtered  shell
554/tcp   filtered  rtsp
1720/tcp  filtered  H.323/Q.931
1723/tcp  filtered  pptp
2000/tcp  filtered  cisco-sccp
3000/tcp  open      ppp
5060/tcp  filtered  sip
7070/tcp  open      realserver
8080/tcp  open      http-proxy
68/udp    open|filtered dhcpc
161/udp   open      snmp
514/udp   open|filtered syslog
518/udp   open|filtered ntalk
1719/udp  open|filtered h323gatestat
5060/udp  open|filtered sip

```

图 2-26 nmap 端口扫描应用实例

2.4.7 nmap 版本侦测

nmap 的版本侦测功能主要用来确定目标主机开放的端口上运行的应用程序及版本信息，nmap 的版本侦测支持 TCP/UDP 协议，支持多种平台的服务侦测，支持 IPV6 功能，并能识别几千种服务签名，下面介绍 nmap 版本侦测的使用方法。

1. 版本侦测的用法

nmap 在版本侦测方面的命令选项非常简单，常用的语法如下：

nmap [选项或参数] 目标主机

nmap 在版本侦测方面的常用选项及含义如表 2-9 所示。

表 2-9 nmap 版本侦测的常用选项及含义

选项	含义
-sV	设置 nmap 进行版本侦测
--version-intensity <level>	设置版本侦测的强度值，取值范围为 0~9，默认是 7。这个数值越高，探测出的服务版本就越精确，但扫描的时间也更长
--version-light	设置使用轻量级的侦测方式，相当于侦测强度值为 2
--version-all	尝试使用所有的 probes 进行侦测，相当于侦测强度为 9
--version-trace	显示版本侦测的详细过程

2. 使用实例

下面以探测“23.76.232.59”主机上运行的应用程序的版本信息为例，简单演示版本探测的使用方法。首先，在联网的服务器上执行如下命令：

```
nmap -sV 23.76.232.59
```

执行结果如图 2-27 所示。

```
root@localhost ~]# nmap -sV 23.76.232.59
Starting Nmap 6.40 ( http://nmap.org ) at 2013-12-21 22:16 CST
Nmap scan report for 23.76.232.59
Host is up (0.0016s latency).
Not shown: 980 closed ports
PORT      STATE SERVICE      VERSION
21/tcp    filtered ftp
22/tcp    open  ssh          OpenSSH 4.3 (protocol 2.0)
80/tcp    open  http         Apache httpd 2.0.63 ((Unix) DAV/2 mod_jk/1.2.14 PHP/5.2.5)
111/tcp   filtered rpcbind
135/tcp   filtered msrpc
445/tcp   filtered microsoft-ds
514/tcp   filtered shell
554/tcp   filtered rtsp
1102/tcp  open  adobeserver-1?
1521/tcp  open  oracle-tns   Oracle TNS Listener 10.2.0.1.0 (for Linux)
1720/tcp  filtered H.323/Q.931
1723/tcp  filtered pptp
2000/tcp  filtered cisco-sccp
3306/tcp  open  mysql        MySQL 5.1.30
4444/tcp  filtered krb524
5060/tcp  filtered sip
8009/tcp  open  ajp13?
8099/tcp  open  unknown
8888/tcp  open  http         Apache Tomcat/Coyote JSP engine 1.1
```

图 2-27 nmap 版本探测实例

从图 2-27 中 nmap 的输出可以看到每个端口对应的服务名称以及详细的版本信息，通过对服务器上运行服务的了解，以及对服务版本的探测，基本能判断出来此服务器是否存在软件漏洞，进而提醒运维管理人员进行端口关闭或升级软件等操作，尽早应对可能出现的安全威胁。

2.4.8 nmap 操作系统探测

操作系统探测主要是对目标主机运行的操作系统类型及版本信息进行检测。nmap 拥有丰富的系统指纹库，目前可以识别近 3000 种操作系统与设备类型。下面介绍 nmap 操作系统探测的使用方法。

1. 操作系统探测的用法

nmap 在操作系统探测方面提供的命令选项比较少，常用的语法如下：

nmap [选项或参数] 目标主机

nmap 在操作系统侦测方面的常用选项及含义如表 2-10 所示。

表 2-10 nmap 操作系统侦测的常用选项及含义

选项	含义
-O	设置 nmap 进行操作系统侦测
--osscan-guess	猜测目标主机的操作系统类型，nmap 会给出可能性的比率，用户可以根据提供的比率综合判断操作系统类型

2. 使用实例

下面以探测“192.168.12.118”和“192.168.12.119”主机的操作系统类型为例，简单演示操作系统侦测的使用方法。首先，在联网的服务器上执行如下命令：

```
nmap -O --osscan-guess 192.168.12.118-119
```

执行结果如图 2-28 所示。

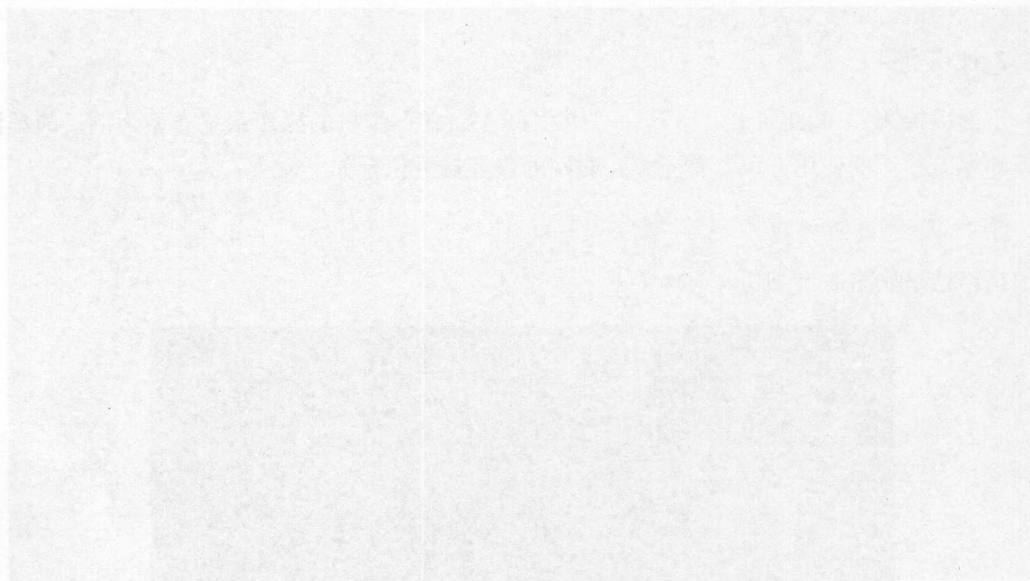
```
[root@localhost ~]# nmap -O --osscan-guess 192.168.12.118-119
Starting Nmap 6.40 ( http://nmap.org ) at 2013-12-21 23:17 CST
Nmap scan report for bogon (192.168.12.118)
Host is up (0.00014s latency).
Not shown: 998 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
8000/tcp   open  http
MAC Address: D4:AE:52:7A:7C:57 (Dell)
Device type: general purpose
Running: Linux 2.6.X
OS CPE: cpe:/o:linux:linux kernel:2.6
OS details: Linux 2.6.9 - 2.6.30
Network Distance: 1 hop

Nmap scan report for 192.168.12.119
Host is up (0.00012s latency).
Not shown: 997 filtered ports
PORT      STATE SERVICE
22/tcp    closed ssh
80/tcp    open  http
443/tcp   open  https
MAC Address: 44:37:E6:90:0D:AC (Hon Hai Precision Ind.Co.Ltd)
Aggressive OS guesses: VMware ESXi Server 5.0 (96%)
                        VMware ESXi Server 4.1 (95%)
                        FreeBSD 7.0-RELEASE-p1 - 10.0-CURRENT (95%)
                        FreeBSD 8.0-BETA2 - 9.1-RELEASE (93%)
                        VMware ESX Server 4.0.1 (93%)
                        FreeBSD 5.2.1-RELEASE (93%)
                        FreeBSD 5.3 or 5.5 (x86) (93%)
                        VMware ESXi Server 4.1.0 (92%)
Network Distance: 1 hop
```

图 2-28 nmap 操作系统侦测实例

从图 2-28 中可以看出，在指定了“-O”选项后，nmap 命令首先执行了主机发现操

作，接着执行了端口扫描操作，然后根据端口扫描的结果进行操作系统类型的侦探，获取到的信息有设备类型、操作系统版本、操作系统的 CPE 描述、操作系统的细节和网络距离。如果不能确定操作系统的版本，会猜测每个系统版本的可能性比率，例如对于“192.168.12.119”主机，nmap 给出最可能的操作系统版本是 VMware ESXi Server 5.0，而事实上，此主机确实安装了这个版本的系统。由此可见，nmap 的操作系统侦测功能是多么的强大。



从图 2-22 可以看到，nmap 对服务器上的操作系统进行了探测，并给出了探测到的操作系统版本信息。

2.4.6 nmap

操作系统版本探测是 nmap 的一个重要功能，nmap 拥有丰富的系统信息，可以用于探测系统漏洞。

图 2-22 显示了 nmap 的操作系统探测结果。

1. 操作系统探测的用途

操作系统探测是 nmap 的一个重要功能，可以用于探测系统漏洞。

nmap 在操作系统探测方面提供了许多功能，可以用于探测系统漏洞。

数据安全工具 DRBD、extundelete

3.1 数据镜像软件 DRBD 介绍

分布式块设备复制 (Distributed Replicated Block Device, DRBD), 是一种基于软件的、基于网络的块复制存储解决方案, 主要用于对服务器之间的磁盘、分区、逻辑卷等进行数据镜像。当用户将数据写入本地磁盘时, 还会将数据发送到网络中另一台主机的磁盘上, 这样本地主机 (主节点) 与远程主机 (备节点) 的数据就可以保证实时同步, 当本地主机出现问题, 远程主机上还保留着一份相同的数据, 可以继续使用, 保证了数据的安全。

3.1.1 DRBD 的基本功能

DRBD 的核心功能就是数据的镜像, 其实现方式是通过网络来镜像整个磁盘设备或磁盘分区, 将一个节点的数据通过网络实时地传送到另一个远程节点, 保证两个节点间数据的一致性, 这有点类似于一个网络 RAID1 的功能。对于 DRBD 数据镜像来说, 它具有如下特点:

- 实时性。当应用对磁盘数据有修改操作时, 数据复制立即发生。
- 透明性。应用程序的数据存储在镜像设备上透明和独立的。数据可以存储在基于网络的不同服务器上。
- 同步镜像。当本地应用申请写操作时, 同时也在远程主机上开始进行写操作。
- 异步镜像。当本地写操作已经完成时, 才开始对远程主机进行写操作。

3.1.2 DRBD 的构成

DRBD 是 Linux 内核存储层中的一个分布式存储系统，具体来说由两部分构成，一部分是内核模板，主要用于虚拟一个块设备；一部分是用户空间管理程序，主要用于和 DRBD 内核模块通信，以管理 DRBD 资源。在 DRBD 中，资源主要包含 DRBD 设备、磁盘配置、网络配置等。

一个 DRBD 系统有两个以上节点构成，分为主用节点和备用节点两个角色，在主用节点上，可以对 DRBD 设备进行不受限制的读写操作，可以用来初始化、创建、挂载文件系统。在备用节点上，DRBD 设备无法挂载，只能用来接收主用节点发送过来的数据，也就是说备用节点不能用于读写访问，这样做的目的是保证数据缓冲区的一致性。

主用节点和备用节点不是固定不变的，可以通过手工方式改变节点的角色，备用节点可以升级为主用节点，同时主用节点也可以降级为备用节点。

DRBD 设备在整个 DRBD 系统中位于物理块设备之上，文件系统之下，在文件系统和物理磁盘之间形成了一个中间层，当用户在主用节点的文件系统中写入数据时，数据被正式写入磁盘前会被 DRBD 系统截获，同时，DRBD 在捕捉到有磁盘写入的操作时，就会通知用户空间管理程序把这些数据复制一份，写入远程主机的 DRBD 镜像，然后存入 DRBD 镜像所映射的远程主机磁盘。图 3-1 详细展示了 DRBD 系统的运行结构。

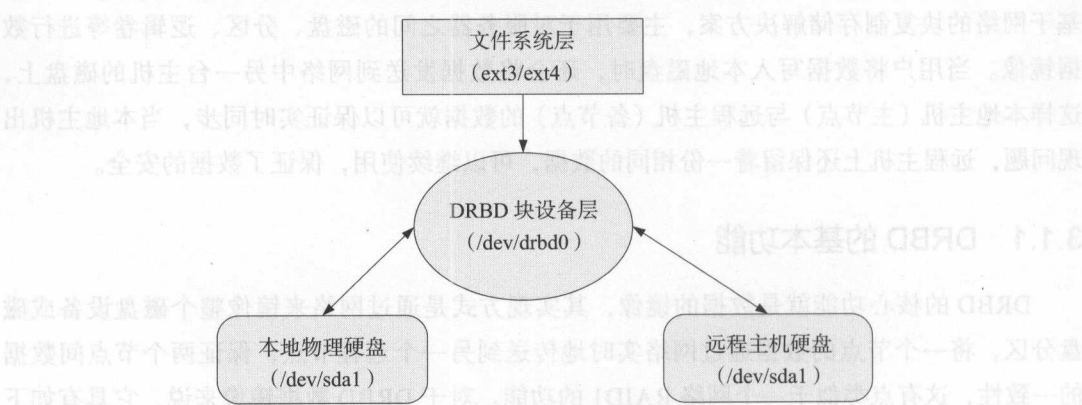


图 3-1 DRBD 系统运行结构

DRBD 负责接收数据，把数据写到本地磁盘，然后发送给另一台主机。另一台主机再将数据存到自己的磁盘中。目前，DRBD 每次只允许对一个节点进行读写访问，这对于通常的故障切换高可用性集群来讲已经足够用了。以后的版本将支持两个节点进行读写存取。

3.1.3 DRBD 与现在的集群的关系

DRBD 由两个或两个以上节点构成，与 HA 集群类似，也有主用节点和备用节点之分，

因而经常用于高可用集群和负载均衡集群系统中作为共享存储设备。由于 DRBD 系统是在 IP 网络中运行, 所以, 在集群中使用 DRBD 作为共享存储设备, 不需要任何硬件投资, 可以节约很多成本, 因为在价格上 IP 网络要比专用的存储网络更经济。

另外, DRBD 也可以用于数据备份、数据容灾等方面。

3.1.4 DRBD 的主要特性

DRBD 系统在实现数据镜像方面有很多有用的特性, 我们可以根据自己的需要和应用环境, 选择适合自己的功能特性。下面依次介绍 DRBD 几个非常重要的应用特性。

1. 单主模式

这是使用最频繁的一种模式, 主要用在高可用集群的数据存储方面, 解决集群中数据共享的问题, 在这种模式下, 集群中只有一个主用节点可以对数据进行读写操作, 可以用在这种模式下的文件系统有 ext3、ext4、xfs 等。

2. 双主模式

这种模式只能在 DRBD8.0 以后的版本中使用, 主要用在负载均衡集群中, 解决数据共享和一致性问题。在这种模式下, 集群中存在两个主用节点, 由于两个主用节点都有可能对数据进行并发的读写操作, 因此单一的文件系统就无法满足需求了, 此时就需要共享的集群文件系统来解决并发读写问题。常用在这个模式下的文件系统有 GFS、OCFS2 等, 通过集群文件系统的分布式锁机制就可以解决集群中两个主用节点同时操作数据的问题。

3. 复制模式

DRBD 提供了三种不同的复制方式, 分别是:

□ 协议 A, 只要本地磁盘写入已经完成, 数据包已经在发送队列中, 则认为一个写操作过程已经完成。

这种方式在远程节点故障或者网络故障时, 可能造成数据丢失, 因为要写入到远程节点的数据可能还在发送队列中。

□ 协议 B, 只要本地磁盘写入已经完成, 并且数据包已经到达远程节点, 则认为一个写操作过程已经完成。

这种方式在远程节点发生故障时, 可能造成数据丢失。

□ 协议 C, 只有本地和远程节点的磁盘已经都确认了写操作完成, 则认为一个写操作过程已经完成。

这种方式没有任何数据丢失, 就目前而言应用最多、最广泛的就是协议 C, 但在此方

式下磁盘的 I/O 吞吐量依赖于网络带宽。建议在网络带宽较好的情况下使用这种方式。

4. 传输完整性校验

这个特性在 DRBD8.2.0 及以后版本中可以使用，DRBD 使用 MD5、SHA-1 或 CRC-32C 等加密算法对信息进行终端到终端的完性验证。利用这个特性，DRBD 对每一个复制到远程节点的数据都生成信息摘要，同时，远端节点也采用同样的方式对复制的数据块进行完整性验证，如果验证信息不对，就请求主节点重新发送。通过这种方式保证镜像数据的完整性和一致性。

5. 脑裂通知和自动修复

由于集群节点间的网络连接临时故障、集群软件管理干预或者人为错误，导致 DRBD 两个节点都切换为主用节点而断开连接，这就是 DRBD 的脑裂问题。发生脑裂意味着数据不能从主用节点复制到备用节点，这样会导致 DRBD 两个节点的数据不一致，并且无法合并。

在 DRBD8.0 及更高版本，实现了裂脑自动修复功能，在 DRBD8.2.1 之后，又实现了裂脑通知特性，在出现脑裂后，一般建议通过手工方式修复脑裂问题，为了彻底解决脑裂问题。在某些情况下脑裂自动修复还是比较可取的，DRBD 自动修复脑裂的策略如下：

- ❑ 丢弃比较新的主用节点所做的修改。在这种模式下，当网络重新建立连接并且发现了脑裂后，DRBD 会丢弃自动切换到主用节点上的主机所修改的数据。
- ❑ 丢弃老的主用节点所做的修改。在这种模式下，DRBD 会丢弃首先切换到主用节点上的主机所修改的数据。
- ❑ 丢弃修改比较少的主用节点的修改。在这种模式下，DRBD 会首先检查两个节点的数据，然后丢弃修改比较少的主机上的数据。
- ❑ 一个节点数据没有发生变化的情况下完美修复脑裂。在这种模式下，如果其中一台主机在发生裂脑时没有发生数据修改，那么就可以完美解决脑裂问题。

3.2 DRBD 的安装与配置

3.2.1 安装环境说明

操作系统统一采用 CentOS5.5-x86-64，安装环境如表 3-1 所示。

表 3-1 DRBD 的安装环境

主机名	IP 地址	镜像磁盘分区
Master-drbd (主用节点)	192.168.12.181	/dev/sdb1
Slave-drbd (备用节点)	192.168.12.182	/dev/sdb1

其中，主用节点和备用节点两块磁盘 /dev/sdb1 是未经格式化的物理磁盘分区，大小均为 10GB。为了不浪费磁盘空间，建议主用节点和备用节点的镜像磁盘大小保持一致。

DRBD 安装的基本拓扑信息如图 3-2 所示。

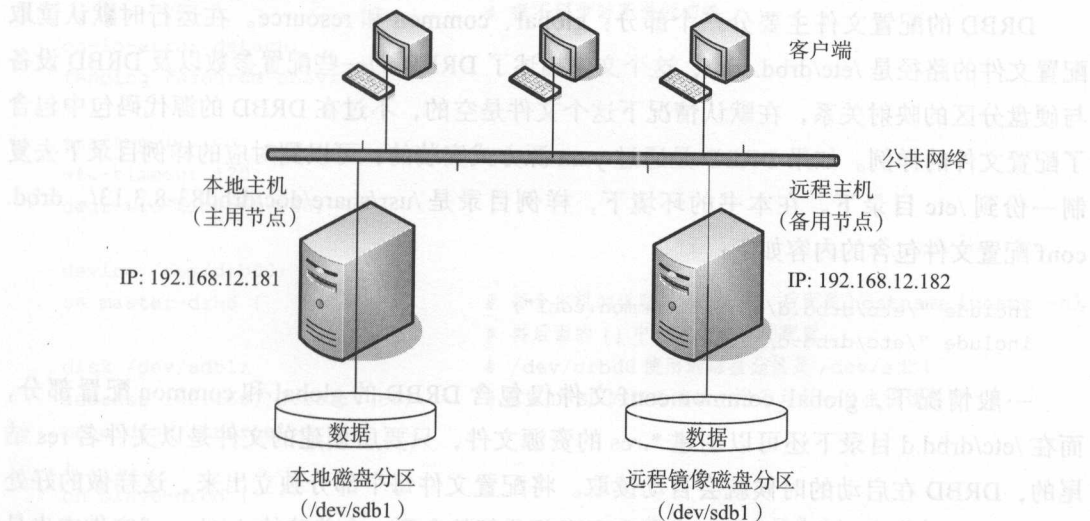


图 3-2 DRBD 安装拓扑

3.2.2 DRBD 的安装部署

DRBD 的安装非常简单，可以通过源码和 yum 源方式进行安装。简单起见，我们直接使用 yum 源方式来安装，其他系统基本类似。下面介绍具体的安装过程。

通过 yum 安装 DRBD 服务，执行如下命令：

```
[root@master-drbd ~]# yum -y install kmod-drbd83 drbd83
```

加载 DRBD 模块到内核：

```
[root@master-drbd ~]# modprobe drbd
```

检查 DRBD 是否安装成功：

```
[root@master-drbd ~]# lsmod | grep -i drbd
drbd                300440  0
```

查看 drbd.ko 安装的路径：

```
[root@master-drbd ~]# modprobe -l | grep -i drbd
/lib/modules/2.6.18-194.el5/weak-updates/drbd83/drbd.ko
```

安装成功之后 drbd 相关工具（drbdadm, drbdsetup）被安装在 /sbin 目录下面，并且会建

立 /etc/init.d/drbd 启动脚本。

3.2.3 快速配置一个 DRBD 镜像系统

DRBD 的配置文件主要分三个部分：global、common 和 resource。在运行时默认读取配置文件的路径是 /etc/drbd.conf，这个文件描述了 DRBD 的一些配置参数以及 DRBD 设备与硬盘分区的映射关系，在默认情况下这个文件是空的，不过在 DRBD 的源代码包中包含了配置文件的样例。如果 DRBD 是通过 yum 源方式安装的，可以到对应的样例目录下去复制一份到 /etc 目录下。在本书的环境下，样例目录是 /usr/share/doc/drbd83-8.3.13/。drbd.conf 配置文件包含的内容如下：

```
include "/etc/drbd.d/global_common.conf";
include "/etc/drbd.d/*.res";
```

一般情况下，global_common.conf 文件仅包含 DRBD 的 global 和 common 配置部分，而在 /etc/drbd.d 目录下还可以创建 *.res 的资源文件，只要所创建的文件是以文件名 res 结尾的，DRBD 在启动的时候就会自动读取。将配置文件每个部分独立出来，这样做的好处是便于管理和维护。其实将 DRBD 所有配置部分都整合到一个单独的 drbd.conf 文件中也是可以的，不过，对于需要配置的资源比较多的情况，这样做就会变得混乱，难于管理。

为了讲述方便，下面我们将 DRBD 的所有配置都集中到一个配置文件中。下面是两台 DRBD 主机节点配置文件 drbd.conf 的简单示例。

```
[root@master-drbd ~]# cat /etc/drbd.conf
# drbd.conf
global {
    usage-count no;          # 是否参加 DRBD 使用者统计，默认参加
}
common {
    syncer { rate 200M; }    # 设置主用节点和备用节点同步时的网络速率最大值，单位是字节
}
resource r0 {               # 资源名字为 r0
    protocol C;              # 使用 DRBD 的第三种同步协议，表示收到远程主机的写入确认后认为写入完成
    handlers {
        pri-on-incon-degr "echo o > /proc/sysrq-trigger ; halt -f";
        pri-lost-after-sb "echo o > /proc/sysrq-trigger ; halt -f";
        local-io-error "echo o > /proc/sysrq-trigger ; halt -f";
        fence-peer "/usr/lib64/heartbeat/drbd-peer-outdater -t 5";
        pri-lost "echo pri-lost. Have a look at the log files. | mail -s 'DRBD Alert' root";
        split-brain "/usr/lib/drbd/notify-split-brain.sh root";
        out-of-sync "/usr/lib/drbd/notify-out-of-sync.sh root";
    }
}
```



```

net {
    cram-hmac-alg "sha1";          # DRBD 同步时使用的验证方式和密码信息
    shared-secret "MySQL-HA";
}
disk {                             # 使用 dpod 功能 (drbd outdate-peer daemon) 保证在数
                                   # 据不同步时不进行切换

on-io-error detach;
fencing resource-only;
}
startup {
    wfc-timeout 120;
    degr-wfc-timeout 120;
}
device /dev/drbd0;
on master-drbd {                  # 每个主机的说明以 on 开头，后面是 hostname (uname -n),
                                   # 再后面的 {} 中为这个主机的配置
    disk /dev/sdb1;                # /dev/drbd0 使用的磁盘分区是 /dev/sdb1
    address 192.168.12.181:7788;    # 设置 DRBD 的监听端口，用于与另一台主机通信
    meta-disk internal;
}
on slave-drbd {
    disk /dev/sdb1;                # /dev/drbd0 使用的磁盘分区是 /dev/sdb1
    address 192.168.12.182:7788;    # 设置 DRBD 的监听端口，用于与另一台主机通信
    meta-disk internal;            # DRBD 的元数据存放方式
}
}

```

将上面这个 drbd.conf 文件分别复制到两台主机的 /etc 目录下。drbd.conf 的配置参数很多，有兴趣的读者可以使用命令“man drbd.conf”来查看了解更多的参数说明。

3.3 DRBD 的管理与维护

3.3.1 启动 DRBD

1. 在两个节点执行

在启动 DRBD 之前，需要分别在两台主机的 hdb1 分区上创建供 DRBD 记录信息的数据块。具体是分别在两台主机上执行：

```

[root@master-drbd ~]# drbdadm create-md r0 或者执行 drbdadm create-md all
[root@slave-drbd ~]# drbdadm create-md r0

```

2. 在两个节点启动服务

接着在两个 drbd 节点启动 DRBD 服务，操作如下：

```
[root@master-drbd ~]# /etc/init.d/drbd start
[root@slave-drbd ~]# /etc/init.d/drbd start
```

最好是在两个节点同时启动 DRBD 服务。

3. 在任意节点查看节点状态

登录任意 drbd 节点，然后执行“cat /proc/drbd”命令，输出结果如下：

```
[root@master-drbd ~]# cat /proc/drbd
0: cs:Connected ro:Secondary/Secondary ds:Inconsistent/Inconsistent C r----
   ns:0 nr:0 dw:0 dr:0 al:0 bm:0 lo:0 pe:0 ua:0 ap:0 ep:1 wo:b oos:2007644
```

对输出的含义解释如下：

- ❑ ro 表示角色信息，第一次启动 drbd 时，两个 drbd 节点默认都处于 Secondary 状态。
- ❑ ds 表示磁盘状态信息，“Inconsistent/Inconsistent”即“不一致/不一致”状态，表示两个节点的磁盘数据处于不一致状态。
- ❑ ns 表示网络发送的数据包信息。
- ❑ dw 表示磁盘写信息。
- ❑ dr 表示磁盘读信息。

4. 设置主用节点

由于默认没有主用节点和备用节点之分，因此需要设置两个主机的主次节点。选择需要设置为主用节点的主机，然后执行如下命令：

```
[root@master-drbd ~]# drbdsetup /dev/drbd0 primary -o
```

也可以执行下面命令：

```
[root@master-drbd ~]# drbdadm -- --overwrite-data-of-peer primary all
```

第一次执行完此命令后，如果需要设置哪个是主用节点，就可以使用另外一个命令：

```
[root@master-drbd ~]# /sbin/drbdadm primary r0 或者 /sbin/drbdadm primary all
```

执行此命令后，开始同步两台机器对应磁盘的数据：

```
[root@master-drbd ~]# cat /proc/drbd
version: 8.3.13 (api:88/proto:86-96)
0: cs:SyncSource ro:Primary/Secondary ds:UpToDate/Inconsistent C r-----
   ns:338640 nr:0 dw:0 dr:346752 al:0 bm:20 lo:1 pe:7 ua:64 ap:0 ep:1wo:b oos:10144232
```

```
[>.....] sync'ed: 3.3% (9904/10236)M
finish: 0:46:26 speed: 3,632 (3,184) K/sec
```

从输出可知:

“ro”状态现在变为“Primary/Secondary”,“ds”状态也变为“UpToDate/Inconsistent”,也就是“实时/不一致”状态。现在数据正在主备两台主机的磁盘间进行同步,且同步进度为3.3%,同步速度3.1Mbit/s左右。

等待片刻,再次查看同步状态,输出如下:

```
[root@master-drbd ~]# cat /proc/drbd
version: 8.3.13 (api:88/proto:86-96)
0: cs:Connected ro:Primary/Secondary ds:UpToDate/UpToDate C r-----
   ns:10482024 nr:0 dw:0 dr:10482024 al:0 bm:640 lo:0 pe:0 ua:0 ap:0 ep:1 wo:b oos:0
```

可以看到同步完成了,并且“ds”状态也变为“UpToDate/UpToDate”,即“实时/实时”状态了。

如果第一次设置主用节点和备用节点时使用“/sbin/drbdadm primary r0”命令,那么会提示如下错误:

```
0: State change failed: (-2) Need access to UpToDate data
Command '/sbin/drbdsetup 0 primary' terminated with exit code 17
```

只要第一次执行上面命令成功,以后就可以用“/sbin/drbdadm primary r0”命令设置主用节点了。

5. 挂载 DRBD 设备

由于 mount 操作只能在主用节点上进行,因此只有设置了主用节点后才能格式化磁盘分区,同时,在两个节点中,同一时刻只能有一台处于 primary 状态,另一台处于 secondary 状态,而处于 secondary 状态的节点上不能挂载 DRBD 设备,要在备用节点上挂载 DRBD 设备,必须停止主用节点的 DRBD 服务或将备用节点角色升级为主用节点。

下面首先将 DRBD 设备格式化为 ext3 文件系统,然后在主用节点挂载,操作如下:

```
[root@master-drbd ~]# mkfs.ext3 /dev/drbd0
[root@master-drbd ~]# mount /dev/drbd0 /mnt
```

完成挂载后,就可以在 /mnt 目录下写数据了,此目录下的数据会自动同步到备用节点上。

3.3.2 测试 DRBD 数据镜像

为了验证 DRBD 的数据镜像功能,我们做一个简单的测试,首先在 DRBD 主用节点上的 /mnt 目录下创建一个 200MB 的文件,操作如下:

```
[root@master-drbd ~]# dd if=/dev/zero of=/mnt/testdrbd.tmp bs=10M count=20
[root@master-drbd ~]# ls -al /mnt/testdrbd.tmp
-rw-r--r-- 1 root root 209715200 Mar 17 14:03 testdrbd.tmp
```

完成操作后,接着在备机上查看文件是否已经同步过去,为了保证数据的一致性,需要首先停止备用节点的 DRBD 服务,操作如下:

```
[root@slave-drbd /]# /etc/init.d/drbd stop
Stopping all DRBD resources: .
[root@slave-drbd /]# mount /dev/sdb1 /mnt
[root@slave-drbd /]# df
Filesystem            1K-blocks      Used Available Use% Mounted on
/dev/mapper/VolGroup00-LogVol00
                        75226176    15156412    56186756    22% /
/dev/sda1              101086       19526       76341    21% /boot
tmpfs                  2025204         0    2025204     0% /dev/shm
/dev/sdb1              10317472    359240     9434132     4% /mnt
[root@slave-drbd /]# cd /mnt
[root@slave-drbdmnt]# ll
total 205020
drwx----- 2 root root    16384 Mar 17 13:58 lost+found
-rw-r--r-- 1 root root 209715200 Mar 17 14:03 testdrbd.tmp
```

可以看到,在主用节点 master-drbd 上产生的文件 testdrbd.tmp 也完整地保存到备用节点 slave-drbd 的镜像磁盘设备上。

测试完毕后,要重新启动备用节点的 DRBD 服务,此时必须先卸载 /dev/sdb1 设备,然后才能成功启动 DRBD 服务。



注意 这里挂载的是 /dev/sdb1 设备,而不是 DRBD 设备,因为 DRBD 设备只有在 DRBD 服务启动的时候才加载到系统中。

3.3.3 DRBD 主备节点切换

在系统维护的时候,或者在高可用集群中,当主用节点出现故障时,就需要将主备节点的角色互换。主备节点切换有两种方式,分别是停止 DRBD 服务切换和正常切换,下面依次介绍。

1. 停止 DRBD 服务切换

关闭主用节点服务,此时挂载的 DRBD 分区就自动在主用节点卸载了,操作如下:

```
[root@master-drbd /]# /etc/init.d/drbd stop
Stopping all DRBD resources:
```


然后查看备用节点的 DRBD 状态：

```
[root@slave-drbd /]# cat /proc/drbd
version: 8.3.13 (api:88/proto:86-96)
0: cs:WfConnection ro:Secondary/Unknown ds:UpToDate/DUnknown C r-----
   ns:0 nr:16 dw:16 dr:0 al:0 bm:0 lo:0 pe:0 ua:0 ap:0 ep:1 wo:b oos:0
```

从输出可以看到，现在主用节点的状态变为“Unknown”，接着在备用节点执行切换命令：

```
[root@slave-drbd ~]# drbdadm primary all
```

此时会出现如下报错信息：

```
2: State change failed: (-7) Refusing to be Primary while peer is not outdated
Command 'drbdsetup 2 primary' terminated with exit code 11
```

因此，必须在备用节点执行如下命令：

```
[root@slave-drbd ~]# drbdsetup /dev/drbd0 primary -o
```

或者

```
[root@slave-drbd ~]# drbdadm -- --overwrite-data-of-peer primary all
```

现在就可以正常切换了。接着查看此节点的状态，信息如下：

```
[root@slave-drbd /]# cat /proc/drbd
version: 8.3.13 (api:88/proto:86-96)
0: cs:WfConnection ro:Primary/Unknown ds:UpToDate/Outdated C r-----
   ns:0 nr:16 dw:16 dr:0 al:0 bm:0 lo:0 pe:0 ua:0 ap:0 ep:1 wo:b oos:0
```

可以看出，原来的备用节点已经处于“Primary”状态了，而原来的主用节点由于 DRBD 服务未启动，还处于“Unknown”状态，在原来的主用节点服务启动后，会自动变为“Secondary”状态，无需在原来主用节点上再次执行切换到备用节点的命令。

最后，在新的主用节点上挂载 DRBD 设备即可完成主备节点的切换：

```
[root@slave-drbd /]# mount /dev/drbd0 /mnt
```

2. 正常切换

首先在主用节点卸载磁盘分区：

```
[root@master-drbd /]# umount /mnt
```

然后执行：

```
[root@master-drbd ~]# drbdadm secondary all
```

如果不执行这个命令，直接在备用节点执行切换到主用节点的命令，会报如下错误：

```
2: State change failed: (-1) Multiple primaries not allowed by confi
Command 'drbdsetup 2 primary' terminated with exit code 11
```

此时查看 master-drbd 节点的 DRBD 状态，信息如下：

```
[root@master-drbd ~]# cat /proc/drbd
version: 8.3.13 (api:88/proto:86-96)
0: cs:Connected ro:Secondary/Secondary ds:UpToDate/UpToDate C r-----
   ns:36 nr:16 dw:52 dr:97 al:2 bm:0 lo:0 pe:0 ua:0 ap:0 ep:1 wo:b oos:0
```

可以看到，两个节点都处于“Secondary”状态了，那么接下来就要指定一个主用节点，在备用节点执行如下命令：

```
[root@slave-drbd ~]# drbdadm primary all
[root@slave-drbd ~]# cat /proc/drbd
version: 8.3.13 (api:88/proto:86-96)
0: cs:Connected ro:Primary/Secondary ds:UpToDate/UpToDate C r-----
   ns:0 nr:36 dw:36 dr:0 al:0 bm:0 lo:0 pe:0 ua:0 ap:0 ep:1 wo:b oos:0
```

至此，主备节点成功切换角色。最后在新的主用节点挂载 DRBD 磁盘分区即可：

```
[root@slave-drbd ~]# mount /dev/drbd0 /mnt
```

3.4 数据恢复软件 extundelete 介绍

作为一名运维人员，保证数据的安全是根本职责，所以在维护系统的时候，要慎之又慎，但是有时难免会出现数据被误删除的情况，在这个时候该如何快速、有效地恢复数据呢？本节我们就来介绍一下 Linux 系统下常用的几个数据恢复工具。

3.4.1 如何使用“rm -rf”命令

在 Linux 系统下，通过命令“rm -rf”可以将任何数据直接从硬盘删除，并且没有任何提示，同时 Linux 下也没有与 Windows 下回收站类似的功能，也就意味着，数据在删除后通过常规的手段是无法恢复的，因此使用这个命令要非常慎重。在使用 rm 命令的时候，比较稳妥的方法是把命令参数放到后面，这样有一个提醒的作用。其实还有一个方法，那就是将要删除的东西通过 mv 命令移动到系统下的 /tmp 目录下，然后写个脚本定期执行清除操作，这样做可以在一定程度上降低误删除数据的危险性。

其实保证数据安全最好的方法是做好备份，虽然备份不是万能的，但是没有备份是万万不行的。任何数据恢复工具都有一定局限性，都不能保证完整地恢复出所有数据，因

此,把备份作为核心,把数据恢复工具作为辅助是运维人员必须坚持的一个准则。

3.4.2 extundelete 与 ext3grep 的异同

在 Linux 下,基于开源的数据恢复工具有很多,常见的有 debugfs、R-Linux、ext3grep、extundelete 等,比较常用的有 ext3grep 和 extundelete,这两个工具的恢复原理基本一样,只是 extundelete 功能更加强大,本节重点介绍 extundelete 的使用方式。

extundelete 是基于 Linux 的一个数据恢复工具,它通过分析文件系统的日志,解析出所有文件的 inode 信息,从而可以恢复 Linux 下主流的 ext3、ext4 文件系统下被误删除的文件。而 ext3grep 仅支持 ext3 文件系统的恢复。在恢复速度上,extundelete 要快很多,因为 extundelete 的恢复机制是扫描 inode 和恢复数据同时进行,并且支持单个文件恢复、单个目录恢复、inode 恢复、block 恢复、完整磁盘恢复等,而 ext3grep 就略显笨拙了,它需要首先扫描完要恢复数据的所有 inode 信息,然后才能开始数据恢复,所以在恢复速度上相对较慢,并且在功能上也不支持目录恢复、时间段恢复等。

3.4.3 extundelete 的恢复原理

在介绍使用 extundelete 进行恢复数据之前,简单介绍下关于 inode 的知识。在 Linux 下可以通过“ls -ld”命令来查看某个文件或者目录的 inode 值,例如查看根目录的 inode 值,可以输入:

```
[root@cloud1 ~]# ls -ld /
2 /
```

由此可知,根目录的 inode 值为 2。

在利用 extundelete 恢复文件时并不依赖特定文件格式,首先 extundelete 会通过文件系统的 inode 信息(根目录的 inode 一般为 2)来获得当前文件系统下所有文件的信息,包括存在的和已经删除的文件,这些信息包括文件名和 inode。然后利用 inode 信息结合日志去查询该 inode 所在的 block 位置,包括直接块、间接块等信息。最后利用 dd 命令将这些信息备份出来,从而恢复数据文件。

3.4.4 安装 extundelete

extundelete 的官方网站是 <http://extundelete.sourceforge.net/>,其目前的稳定版本是 extundelete-0.2.4。在安装 extundelete 之前需要安装 e2fsprogs 和 e2fsprogs-libs 两个依赖包。

e2fsprogs 和 e2fsprogs-libs 安装非常简单,这里不做介绍。下面是 extundelete 的编译安装过程:

```
[root@cloud1 app]# tar jxvf extundelete-0.2.4.tar.bz2
[root@cloud1 app]# cd extundelete-0.2.4
[root@cloud1 extundelete-0.2.4]# ./configure
[root@cloud1 extundelete-0.2.4]# make
[root@cloud1 extundelete-0.2.4]# make install
```

成功安装 extundelete 后，会在系统中生成一个 extundelete 可执行文件。extundelete 的使用非常简单，读者可以通过“extundelete --help”获得此软件的使用方法。

3.4.5 extundelete 用法详解

extundelete 安装完成后，就可以执行数据恢复操作了，本节详细介绍 extundelete 每个参数的含义。extundelete 用法如下：

```
extundelete --help
```

命令格式：

```
extundelete [options] [action] device-file
```

其中，参数 (options) 有：

- ❑ --version, [-vV]，显示软件版本号。
- ❑ --help，显示软件帮助信息。
- ❑ --superblock，显示超级块信息。
- ❑ --journal，显示日志信息。
- ❑ --after dtime，时间参数，表示在某段时间之后被删的文件或目录。
- ❑ --before dtime，时间参数，表示在某段时间之前被删的文件或目录。

动作 (action) 有：

- ❑ --inode ino，显示节点“ino”的信息。
- ❑ --block blk，显示数据块“blk”的信息。
- ❑ --restore-inode ino[,ino,...]，恢复命令参数，表示恢复节点“ino”的文件，恢复的文件会自动放在当前目录下的 RESTORED_FILES 文件夹中，使用节点编号作为扩展名。
- ❑ --restore-file 'path'，恢复命令参数，表示将恢复指定路径的文件，并把恢复的文件放在当前目录下的 RECOVERED_FILES 目录中。
- ❑ --restore-files 'path'，恢复命令参数，表示将恢复在路径中已列出的所有文件。
- ❑ --restore-all，恢复命令参数，表示将尝试恢复所有目录和文件。
- ❑ -j journal，表示从已经命名的文件中读取扩展日志。
- ❑ -b blocknumber，表示使用之前备份的超级块来打开文件系统，一般用于查看现有超

级块是不是当前所要的文件。

- **-B blocksize**, 通过指定数据块大小来打开文件系统, 一般用于查看已经知道大小的文件。

3.5 实战: extundelete 恢复数据的过程

在数据被误删除后, 第一时间要做的是卸载被删除数据所在的磁盘或磁盘分区, 如果是系统根分区的数据遭到误删除, 就需要将系统进入单用户, 并且将根分区以只读模式挂载。这样做的原因很简单, 因为将文件删除后, 仅仅是将文件的 inode 结点中的扇区指针清零, 实际文件还存储在磁盘上, 如果磁盘以读写模式挂载, 这些已删除的文件的数据块可能被操作系统重新分配出去, 在这些数据块被新的数据覆盖后, 这些数据就真的丢失了, 恢复工具也无力回天。所以, 以只读模式挂载磁盘可以尽量降低数据块中数据被覆盖的风险, 以提高恢复数据成功的比率。

3.5.1 通过 extundelete 恢复单个文件

1. 模拟数据误删除环境

在演示通过 extundelete 恢复数据之前, 我们首先要模拟一个数据误删除环境, 这里我们以 ext3 文件系统为例, 在 ext4 文件系统下的恢复方式与此完全一样。简单的模拟操作过程如下:

```
[root@cloud1 ~]# mkdir /data
[root@cloud1 ~]# mkfs.ext3 /dev/sdc1
[root@cloud1 ~]# mount /dev/sdc1 /data
[root@cloud1 ~]# cp /etc/passwd /data
[root@cloud1 ~]# cp -r /app/ganglia-3.4.0 /data
[root@cloud1 ~]# mkdir /data/test
[root@cloud1 ~]# echo "extundelete test" > /data/test/mytest.txt
[root@cloud1 ~]# cd /data
[root@cloud1 data]# md5sum passwd
0715baf8f17a6c51be63b1c5c0fbe8c5 passwd
[root@cloud1 data]# md5sum test/mytest.txt
eb42e4b3f953ce00e78e11bf50652a80 test/mytest.txt
[root@cloud1 data]# rm -rf /data/*
```

2. 卸载磁盘分区

在将数据误删除后, 立刻需要做的就是卸载这块磁盘分区:

```
[root@cloud1 data]# cd /mnt
[root@cloud1 mnt]# umount /data
```

3. 查询可恢复的数据信息

通过 `extundelete` 命令可以查询 `/dev/sdc1` 分区可恢复的数据信息：

```
[root@cloud1 /]# extundelete /dev/sdc1 --inode 2
.....
File name | Inode number | Deleted status
.         | 2             | Deleted
..        | 2             | Deleted
lost+found | 11            | Deleted
passwd    | 49153         | Deleted
test      | 425985        | Deleted
ganglia-3.4.0 | 245761       | Deleted
```

根据上面的输出，标记为 `Deleted` 状态的是已经删除的文件或目录。同时还可以看到每个已删除文件的 `inode` 值，接下来就可以恢复文件了。

4. 恢复单个文件

执行如下命令开始恢复文件：

```
[root@cloud1 /]# extundelete /dev/sdc1 --restore-file passwd
Loading filesystem metadata ... 40 groups loaded.
Loading journal descriptors ... 54 descriptors loaded.
Successfully restored file passwd
[root@cloud1 /]# cd RECOVERED_FILES/
[root@cloud1 RECOVERED_FILES]# ls
passwd
[root@cloud1 RECOVERED_FILES]# md5sum passwd
0715baf8f17a6c51be63b1c5c0f8e8c5 passwd
```

`extundelete` 恢复单个文件的参数是“`--restore-file`”，这里需要注意的是，“`--restore-file`”后面指定的是恢复文件路径，这个路径是文件的相对路径。相对路径是相对于原来文件的存储路径而言的，比如，原来文件的存储路径是 `/data/passwd`，那么在参数后面直接指定 `passwd` 文件即可，如果原来文件的存储路径是 `/data/test/mytest.txt`，那么在参数后面通过“`test/mytest.txt`”指定即可。

在文件恢复成功后，`extundelete` 命令默认会在执行命令的当前目录下创建一个 `RECOVERED_FILES` 目录，此目录用于存放恢复的文件，所以执行 `extundelete` 命令的当前目录必须是可写的。

根据上面的输出，通过 `md5sum` 命令校验，校验码与之前的完全一致，表明文件恢复成功。

3.5.2 通过 extundelete 恢复单个目录

extundelete 除了支持恢复单个文件，也支持恢复单个目录，在需要恢复目录时，通过“--restore-directory”选项即可恢复指定目录的所有数据。

继续在上面模拟的误删除数据环境下操作，现在要恢复 /data 目录下的 ganglia-3.4.0 文件夹，操作如下：

```
[root@cloud1 mnt]# extundelete /dev/sdcl --restore-directory /ganglia-3.4.0
Loading filesystem metadata ... 40 groups loaded.
Loading journal descriptors ... 247 descriptors loaded.
Searching for recoverable inodes in directory /ganglia-3.4.0 ...
781 recoverable inodes found.
Looking through the directory structure for deleted files ...
4 recoverable inodes still lost.
[root@cloud1 mnt]# ls
RECOVERED_FILES
[root@cloud1 mnt]# cd RECOVERED_FILES/
[root@cloud1 RECOVERED_FILES]# ls
ganglia-3.4.0
```

可以看到之前删除的目录 ganglia-3.4.0 已经成功恢复了，进入这个目录检查发现：所有文件内容和大小都正常。

3.5.3 通过 extundelete 恢复所有误删除数据

当需要恢复的数据较多时，一个个地指定文件或目录将是一项非常繁重和耗时的工作，不过，extundelete 考虑到了这点，此时可以通过“--restore-all”选项来恢复所有被删除的文件或文件夹。

仍然在上面模拟的误删除数据环境下操作，现在要恢复 /data 目录下所有数据，操作过程如下：

```
[root@cloud1 mnt]# extundelete /dev/sdcl --restore-all
Loading filesystem metadata ... 40 groups loaded.
Loading journal descriptors ... 247 descriptors loaded.
Searching for recoverable inodes in directory / ...
781 recoverable inodes found.
Looking through the directory structure for deleted files ...
0 recoverable inodes still lost.
[root@cloud1 mnt]# ls
RECOVERED_FILES
[root@cloud1 mnt]# cd RECOVERED_FILES/
[root@cloud1 RECOVERED_FILES]# ls
ganglia-3.4.0 passwd test
[root@cloud1 RECOVERED_FILES]# du -sh /mnt/RECOVERED_FILES/*
```

```
15M      /mnt/RECOVERED_FILES/ganglia-3.4.0
4.0K     /mnt/RECOVERED_FILES/passwd
8.0K     /mnt/RECOVERED_FILES/test
```

可以看到所有数据全部完整地恢复了。

3.5.4 通过 extundelete 恢复某个时间段的数据

有时候删除了大量的数据量，其中很多数据都是没用的，我们仅需要恢复其中的一部分数据，此时，如果采用恢复全部数据的办法，不但耗时，而且浪费资源，在这种情况下，就需要采用另外一种恢复机制有选择地恢复，extundelete 提供了 “--after” 和 “--before” 参数，可以通过指定某个时间段，进而只恢复这个时间段内的数据。

下面通过一个简单示例描述如何恢复某个时间段内的数据。

首先假定在 /data 目录下有个刚刚创建的压缩文件 ganglia-3.4.0.tar.gz，然后删除此文件，接着卸载 /data 分区，开始恢复一小时内的文件，操作如下：

```
[root@cloud1 ~]# cd /data/
[root@cloud1 data]# cp /app/ganglia-3.4.0.tar.gz /data
[root@cloud1 data]# date +%s
1379150309
[root@cloud1 data]# rm -rf ganglia-3.4.0.tar.gz
[root@cloud1 data]# cd /mnt
[root@cloud1 mnt]# umount /data
[root@cloud1 mnt]# date +%s
1379150340
[root@cloud1 mnt]# extundelete --after 1379146740 --restore-all /dev/sdc1
Only show and process deleted entries if they are deleted on or after 1379146740
and before 9223372036854775807.
Loading filesystem metadata ... 40 groups loaded.
Loading journal descriptors ... 247 descriptors loaded.
Searching for recoverable inodes in directory / ...
779 recoverable inodes found.
[root@cloud1 mnt]# cd RECOVERED_FILES/
[root@cloud1 RECOVERED_FILES]# ls
ganglia-3.4.0.tar.gz
```

可以看到，刚才删除的文件已经成功恢复，而在 /data 目录下还有很多被删除的文件却没有恢复，这就是 “--after” 参数控制的结果，因为 /data 目录下其他文件都是在一天之前删除的，而我们恢复的是一个小时之内被删除的文件，这就是没有恢复其他被删除文件的原因。

在这个操作过程中，需要注意的是 “--after” 参数后面跟的时间是个总秒数。起算时间为 “1970-01-01 00:00:00 UTC”，通过 “date +%s” 命令即可将当前时间转换为总秒数，因为恢复的是一个小时之内的数据，所以 “1379146740” 这个值就是通过 “1379150340” 减去 “60*60=3600” 获得的。

第二部分 Part 2

运维故障排查篇

4.1 Linux 系统故障的处理思路

处理系统故障是 Linux 系统运维工作中最基础的一项。作为合格的 Linux 运维人员，一定要有一套清晰、明确的解决故障思路，这样在故障出现时才能迅速定位并解决问题。本章重点介绍处理 Linux 系统故障的基本思路和多里方法。

Linux 系统下的故障是千变万化、多种多样的，每个问题的现象都不尽相同，解决方法也各有异同，因此不可能把每种问题和解决方法都一一介绍。但是这些问题的解决方法却有一个标准的思路，俗话说万变不离其宗，掌握了解决故障的思路，相信一切问题都会迎刃而解。

这里给出处理问题的思路：

- 第4章 Linux系统运维故障排查思路
- 第5章 Linux故障排查案例实战

1) 重视报错提示信息。报错提示信息是故障排查的第一线索。一般情况下这个提示信息基本定位了问题的所在。如果忽视了报错提示信息，一些错误信息视而不见，问题永远得不到解决。

2) 查阅日志文件。有时候报错提示信息给出了问题的表面现象，要想更深入地了解问题，必须查看相应的日志文件。日志文件主要分为系统日志文件（/var/log）和应用程序日志文件。结合这两个日志文件，一般就能定位问题所在。

3) 分析、定位问题。这个步骤相对比较复杂的，根据报错信息，结合日志文件，同时还要考虑其他相关情况，最终找到产生问题的原因。

4) 解决问题。找到了问题产生的原因，解决问题就是很简单的事情了。

Linux 系统运维故障排查思路

4.1 Linux 系统故障的处理思路

处理系统故障是 Linux 系统运维工作中最基础的一项。作为一名合格的 Linux 运维人员，一定要有一套清晰、明确的解决故障思路，这样在问题出现时才能迅速定位并解决问题，本章重点介绍处理 Linux 系统故障的基本思路和常见方法。

Linux 系统下的故障是千变万化、多种多样的，每个问题的现象都不尽相同，解决问题的方法也各有异同，因此不可能把每种问题和解决方法都一一介绍，但是这些问题的解决方法却有一个标准的思路，俗话说万变不离其宗，掌握了解决问题的思路，相信一切问题都会迎刃而解。

这里给出处理问题的一般思路：

- 1) 重视报错提示信息。每当错误出现，都会给出错误提示信息，一般情况下这个提示基本定位了问题的所在，因此一定要重视这个报错信息，如果对这些错误信息视而不见，问题永远得不到解决。

- 2) 查阅日志文件。有时候报错信息只给出了问题的表面现象，要想更深入地了解问题，必须查看相应的日志文件，而日志文件又分为系统日志文件（/var/log）和应用程序日志文件，结合这两个日志文件，一般就能定位问题所在。

- 3) 分析、定位问题。这个过程是比较复杂的，根据报错信息，结合日志文件，同时还要考虑其他相关情况，最终找到产生问题的原因。

- 4) 解决问题。找到了问题出现的原因，解决问题就是很简单的事情了。

从这个流程可以看出，解决问题的过程就是分析、查找问题的过程，一旦确定问题产生的原因，故障也就随之解决了。

4.2 Linux 系统无法启动的解决方法

这是 Linux 系统最常见的故障，系统在掉电，以及执行配置更新、软件升级、内核升级后都有可能导致无法正常启动，究其原因，有很多种，常见有如下几种。

1) 文件系统破坏，一般是 Linux 的根分区文件系统遭到破坏，导致系统无法启动，这种情况一般是由系统突然掉电或者非法关机引起的。

2) 文件系统配置不当，比如 `/etc/inittab` 文件、`/etc/fstab` 文件等配置错误或丢失，导致系统错误，无法启动。这种情况一般是执行配置更新时人为导致的。

3) Linux 内核文件丢失或崩溃，从而导致 Linux 系统无法启动，这种情况可能是由于内核升级错误或者内核存在 bug 引起的。

4) 系统引导程序出现问题，比如 `grub` 丢失或者损坏，导致系统无法引导启动。这种情况一般是由人为修改错误或者文件系统故障导致的。

5) 系统硬件故障，比如主板、电源、硬盘等出现问题，导致 Linux 系统无法启动。这种情况基本都是由于服务器硬件问题导致的。

从上面列出的常见故障可知，导致系统无法启动的问题主要有两个：硬件原因和操作系统原因。对于由于硬件导致的问题，只需通过更换硬件设备即可解决，而由于操作系统导致的问题，虽然问题可能各不相同，但是在多数情况下都可以用相对简单、统一的一些方法来恢复系统。下面就针对上面提出的几个问题，结合 CentOS 系统环境，给出一些常用的、普遍的解决系统无法启动的方法。

4.2.1 文件系统破坏导致系统无法启动

当前 Linux 发行版普遍采用的都是 `ext3`、`ext4` 文件系统，而这两种文件系统都是具有日志记录功能的日志文件系统，并且可以进行简单的容错和纠错。了解日志文件系统的读者都知道，日志文件系统并不是把数据实时写到磁盘，而是定期批量写入磁盘，但是文件系统的所有读写操作都会实时记录到日志文件中，当系统发生掉电等错误导致数据没有写入磁盘时，可以通过日志文件中的记录，回滚发生故障时的读写操作，进而保证数据和文件系统的一致性，但是由于实际环境的复杂性和应用的多样性，文件系统的容错机制并不能保证每次都能自我修复成功，此时就需要运维人员介入进行手动修复。

下面这个示例演示的就是这样的情况：Linux 无法自动修复错误的文件系统，然后自动进入了单用户模式下或者出现一个交互界面，提示用户介入手动修复。

```

checking root filesystem
/dev/sda6 contains a file system with errors, check forced
/dev/sda6:
Unattached inode 1882681693
/dev/sda6: UNEXPECTED INCONSISTENCY; RUN fsck MANUALLY
(i.e., without -a or -p options)
FAILED
/contains a file system with errors check forced
anerror occurred during the file system check
****dropping you to a shell;the system will reboot
****when you leave the shell
Press enter for maintenance
(or type Control-D to continue):
give root password for maintenance

```

从这个错误可以看出，操作系统 /dev/sda6 分区文件系统出现了问题，这个问题发生的机率很高，通常引起这个问题的原因主要是系统突然掉电，引起文件系统结构不一致。一般情况下解决此问题的办法是采用 fsck 命令，进行强制修复。

根据上面的错误提示，按下 Control+D 组合键后系统自动重启，输入 root 密码后进入系统修复模式，在修复模式下，可以执行 fsck 命令，具体操作过程如下：

```

[root@localhost /]# umount /dev/sda6
[root@localhost /]# fsck.ext3 -y /dev/sda6
e2fsck 1.39 (29-May-2006)
/ contains a file system with errors, check forced.
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Inode 1882681693 ref count is 2, should be 1. Fix<y>? yes
Unattached inode 1882681693
Connect to /lost+found<y>? yes
Inode 1882681693 ref count is 2, should be 1. Fix<y>? yes
Pass 5: Checking group summary information
Block bitmap differences: -(519--529) -9273
Fix<y>? yes
.....
/: ***** FILE SYSTEM WAS MODIFIED *****
/: 19/128520 files (15.8% non-contiguous), 46034/514048 blocks

```

需要说明的是，在修复文件系统时，一定要先卸载故障的磁盘分区，然后再执行 fsck 命令进行修复。

在修复的过程中，可以看到文件系统中哪些 inode 有问题，无法恢复的数据会存放在文件系统下的 lost+found 目录中。只要数据丢失不是很严重，修复完成后一般都能成功启动系统。

4.2.2 /etc/fstab 文件丢失导致系统无法启动

熟悉 Linux 的读者都知道, /etc/fstab 文件中存放了系统的文件系统挂载的相关信息, 如果正确地配置了该文件, 那么在 Linux 启动时, 系统会读取此文件, 自动挂载 Linux 的各个分区, 如果此文件配置错误, 或者丢失, 就会导致系统无法启动, 具体的故障现象是在检测 mount partition 时出现:

```
starting system logger
```

此后系统启动就停止了。

针对这个问题, 常见的思路就是想办法恢复 /etc/fstab 这个文件的信息, 只要恢复了此文件, 系统就能自动挂载每个分区并正常启动。可能很多读者首先想到的是将系统切换到单用户模式下, 然后手动挂载分区, 最后结合系统信息, 重建 /etc/fstab 文件。

但是这种方法是行不通的, 因为 fstab 文件丢失导致 Linux 无法挂载任何一个分区, 即使 Linux 还能切换到单用户模式下, 所以此时的系统也只是一个 read-only 的文件系统, 无法向磁盘写入任何信息, 那么重建 /etc/fstab 文件也无法实现。

有没有其他方法可以实现重建 /etc/fstab 文件? 当然有! 下面我们就介绍另外一种方法, 利用 linux rescue 修复模式登录系统, 进而获取分区和挂载点信息, 重构 /etc/fstab 文件。这里以 CentOS 5.8 为例进行介绍。

首先将系统光盘放入光驱, 设置 BIOS 从光驱启动, 这样系统就从光驱引导, 然后在 boot 后输入: linux rescue, 如图 4-1 所示。

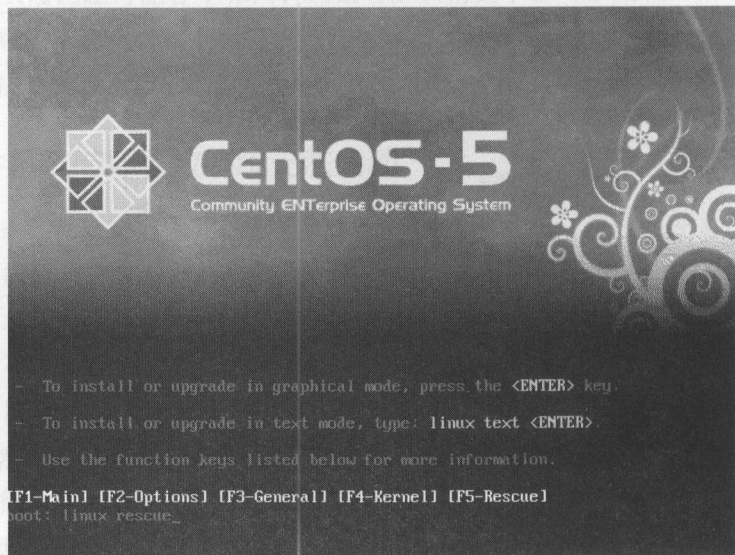


图 4-1 设置 Linux 进入修复模式

接着系统开始自动引导，进入图 4-2 所示画面。这里是选择模式使用的语言，可以按照自己的需要设定。在这里我们选择“English”，然后按 tab 键，选中“OK”，按回车键进入下一步。

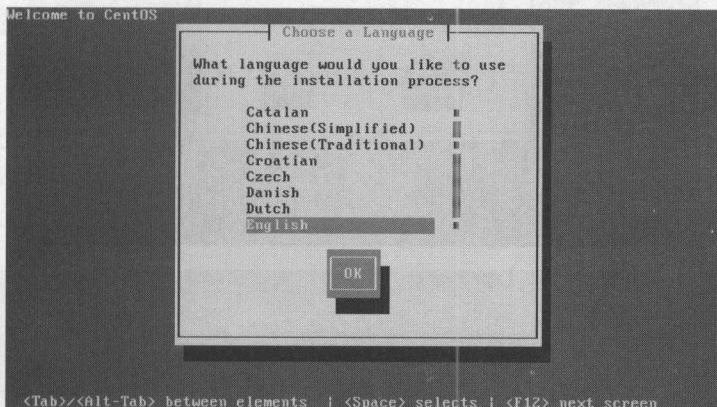


图 4-2 选择语言

下面进入的是键盘选择界面，如图 4-3 所示，这里选择默认的“us”即可。

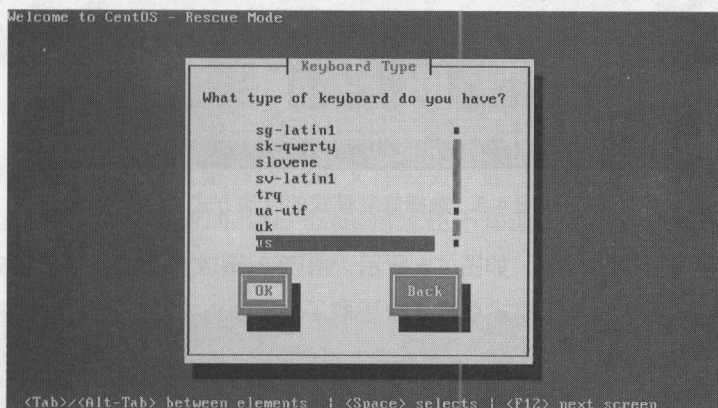


图 4-3 选择键盘类型

下面进入是否启用网络界面，如图 4-4 所示。由于系统已经无法启动，我们已经在 Linux 系统上进行操作了，启用网络与否都无所谓。这里选择不启用，即“No”。

接下来进入最关键的步骤，如图 4-5 所示，修复模式会自动将系统的所有分区挂载到 /mnt/sysimage 目录下，选择“Continue”，则修复环境进入 read-write 状态下，可以对分区进行读写操作；选择“Read-Only”，修复环境进入只读模式。由于我们要重建 fstab 文件到 /etc 目录下，因此选择“Continue”进入可读写模式下。

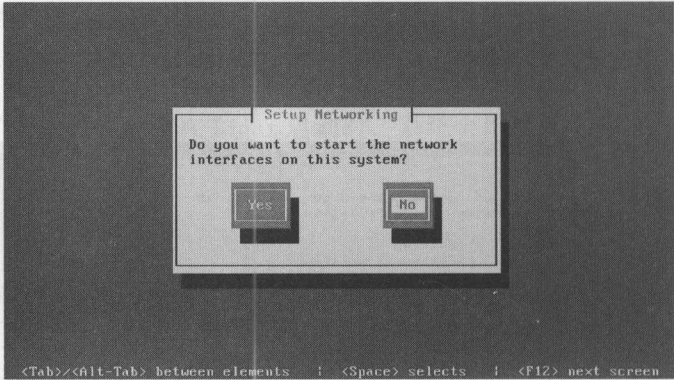


图 4-4 是否启用网络

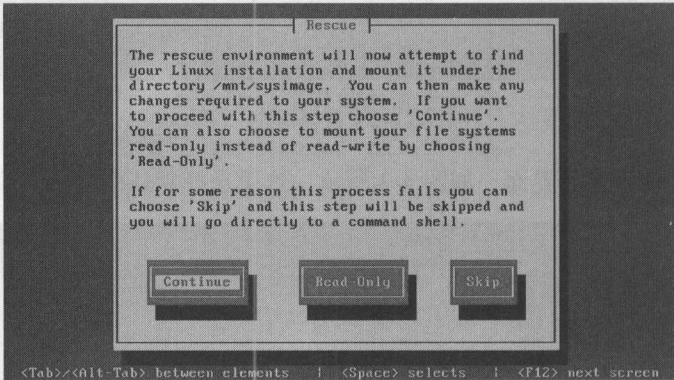


图 4-5 选择修复模式的启动方式

接下来是一个友情提示界面，如图 4-6 所示，由于 fstab 文件丢失，修复模式找不到任何可挂载的分区，从这里可知，修复模式在这里也读取了 /etc/fstab 文件，按回车键进入下一步。

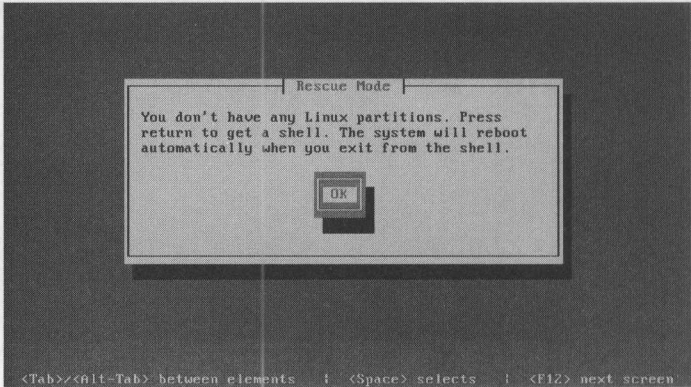


图 4-6 系统无法挂载任何分区

接下来进入修复环境下，可以进行操作了。如图 4-7 所示：

```
Your system is mounted under the /mnt/sysimage directory.
When finished please exit from the shell and your system will reboot.

sh-3.2#
```

图 4-7 修复模式下的命令行

到这里为止，已经可以在修复模式下进行恢复操作了。接下来通过一个示例介绍下恢复 /etc/fstab 文件的详细过程。

下面是一个 /etc/fstab 文件丢失后启动到恢复模式下的系统，分区信息如图 4-8 所示。

```
sh-3.2# df
Filesystem            1K-blocks    Used Available Use% Mounted on
/dev                  334096         0    334096   0% /dev
/tmp/loop0           112576    112576         0 100% /mnt/runtime

sh-3.2# fdisk -l

Disk /dev/sda: 32.6 GB, 32663142400 bytes
255 heads, 63 sectors/track, 3971 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start   End  Blocks   Id  System
/dev/sda1 *          1     13     104391   83  Linux
/dev/sda2            14     535    4192965   82  Linux swap
/dev/sda3           536    3971    27599670   83  Linux

sh-3.2#
```

图 4-8 恢复模式下的磁盘分区信息

由于分区并没有损坏，仅仅是 /etc/fstab 文件丢失，通过 fdisk 命令可以查看到系统分区的完整信息，但是每个分区对应的 label name 信息还不知道，不过可以通过 e2label 命令查看每个分区对应的 label name，如图 4-9 所示。

```
sh-3.2# e2label /dev/sda1
/boot
sh-3.2# e2label /dev/sda3
/
sh-3.2#
```

图 4-9 通过 e2label 命令查看分区的 label name

通过 e2label 命令，可以得到所有分区的挂载点信息，接下来就可以构造出一个 fstab 文件了。

这里有个小技巧：可以参考其同类型系统中 fstab 文件的格式，结合本系统的分区和挂载点信息，构造出一个 fstab 文件。

由于 fstab 文件是存放在系统根目录下的，所以首先需要挂载原来系统的根分区，从上面内容可知根分区对应的设备名为 /dev/sda3，接着在修复模式下创建一个挂载点，然后挂载原来系统的根分区。操作过程如图 4-10 所示。


```
sh-3.2# pwd
/
sh-3.2# mkdir temp
sh-3.2# mount /dev/sda3 /temp
sh-3.2# df
Filesystem            1K-blocks      Used Available Use% Mounted on
/dev                   334896         0    334896   0% /dev
/tmp/loop0             112576       112576         0 100% /mnt/runtime
/dev/sda3              26735280    4453788    20901512   18% /temp
sh-3.2#
```

图 4-10 在修复模式下挂载磁盘分区

通过这种方式将原有根分区的文件全部挂载到 /temp 目录下，接着就可以创建我们需要的 fatab 文件了。恢复完成的 fstab 文件内容如图 4-11 所示。

```
sh-3.2# more /temp/etc/fstab
LABEL=/                /                ext3    defaults    1 1
LABEL=/boot            /boot            ext3    defaults    1 2
tmpfs                  /dev/shm         tmpfs   defaults    0 0
devpts                 /dev/pts         devpts  gid=5,mode=620 0 0
sysfs                  /sys             sysfs   defaults    0 0
proc                   /proc            proc    defaults    0 0
LABEL=SWAP-sda2        swap             swap    defaults    0 0
```

图 4-11 手动重建 fstab 文件

配置完成后，重启系统，看系统能否正常启动。

4.3 Linux 系统无响应（死机）问题分析

Linux 服务器在长期运行后，难免出现无响应现象，俗称“死机”。在系统死机后，屏幕一般会输出故障信息，键盘失去响应，这种情况的常见处理办法就是重启系统，不过在重启前，要重点关注下屏幕的输出信息，因为其提示的可能是引起死机的主要原因，对解决问题有很大帮助。其实还有另一种方法，就是通过串口直连线连接客户机和服务器，将服务器的出错详细信息发送到客户机上。

引起服务器死机的原因有很多，但主要有两个方面：软件问题和硬件问题。下面总结了造成 Linux 系统死机的常见原因和解决问题的思路。

- 1) 系统硬件问题，主要是由 SCSI 卡、主板、RAID 卡、HBA 卡、网卡、硬盘等硬件设备导致的。在这种情况下需要定位硬件故障细节，通过更换硬件来解决问题。
- 2) 外围硬件问题，主要是网络问题导致的。此时就需要从网络设备、网络参数等方面查找和解决问题。
- 3) 软件问题，主要是系统内核 bug、应用软件 bug、驱动程序 bug 等。在这种情况下就需要从升级内核、修复程序 bug、更新驱动程序等方面来解决问题。
- 4) 系统设置问题：主要是系统参数设置不当导致的，可以通过恢复系统到默认状态，关闭防火墙等方面来解决问题。

4.4 Linux 下常见网络故障的处理思路

据统计, Linux 系统下产生的故障有 60% 来自网络方面, 40% 来自系统本身, 可见熟练解决 Linux 下的网络故障, 对 Linux 运维工作有着巨大的帮助。

解决 Linux 网络问题的顺序应该是首先从 Linux 操作系统自身的底层网络开始, 然后逐步向外扩展, 由点及面。下面给出解决网络问题的一般流程。

1) 网络硬件问题, 可以通过检查网线、网卡、集线器、路由器、交换机等是否正常来确认是否由硬件问题造成网络故障。

2) 检查网卡能否正常工作, 可以从网卡驱动是否正常加载、网卡 IP 设置是否正确、系统路由是否设置正确等三个方面进行确认。

3) 检查局域网主机之间联机是否正常, 可以通过 ping 自身 IP、ping 局域网其他主机 IP、ping 网关地址等方式来确认。

4) 检查 DNS 是否设定正确, 可以从 Linux 的 DNS 客户端配置文件 `/etc/resolv.conf`、本地主机文件 `/etc/hosts` 进行确认。

5) 服务是否正常打开, 可以通过 `telnet` 或 `netstat` 命令检测服务是否开启。

6) 检查访问权限是否打开, 可以从本机 `iptables` 防火墙、Linux 内核强制访问控制策略 SELinux 两方面入手。

接下来就针对上面给出的解决网络问题的一般思路, 详细展开讲述。

4.4.1 检查网络硬件问题

检查网络故障, 首先要排除的是网络硬件设备是否存在问题, 比如网线、网卡、集线器、路由器、交换机等是否正常, 这些是网络正常运行的基本条件, 如果发现某些设备出现故障, 只需更换硬件即可解决问题。

4.4.2 检查网卡是否正常工作

(1) 检查网卡是否正常加载

通过 `lsmod`、`ifconfig` 命令可以判断网卡是否正常加载, 如果通过 `ifconfig` 可以显示网络接口 (`eth0`、`eth1` 等) 的配置信息, 表示系统已经找到网卡驱动程序, 检测到网络设备, 网卡加载正常。

(2) 检查网卡 IP 设置是否正确

接下来就要检查网卡的软件设定, 比如 IP 是否配置、配置是否正确、确保 IP 的配置和局域网其他服务器的配置没有冲突。

(3) 检查系统路由表信息是否正确

检查系统路由表状态是处理网络问题的一种很重要方法,下面通过一个简单的例子来阐述这个问题。

假如某台服务器有两块网卡,eth0 的 IP 地址为 10.10.1.239,网关为 10.10.1.254,eth1 的 IP 地址为 192.168.200.30,网关为 192.168.200.1,eth0 通过映射的方式对外提供 SSH 连接服务,而 eth1 仅供局域网主机之间共享数据使用。现在的问题是,外界无法通过 SSH 服务远程登录到此系统,而网卡加载没有问题,网卡 IP 设置也没问题,接下来看看此系统的路由设置:

```
[root@webserver ~]# route
```

```
Kernel IP routing table
```

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
10.10.1.0	*	255.255.255.0	U	0	0	0	eth0
192.168.200.0	*	255.255.255.0	U	0	0	0	eth1
default	192.168.200.1	0.0.0.0	UG	0	0	0	eth1

到这里,问题已经基本排查出来了。

从 route 的输出可知,这个服务器的默认路由是 192.168.200.1,绑定在 eth1 网卡上,而 192.168.200 段的 IP 仅仅供局域网主机之间共享数据使用,没有对外连接的访问权限,因而,外界无法连接到 Linux 系统,也是理所当然的事情了。

定位了问题,解决方法很简单,删除 192 段的默认路由,在 eth0 网卡上添加 10 段的默认路由即可,具体操作如下:

```
[root@webserver ~]# route delete default
```

```
[root@webserver ~]# route add default gw 10.10.1.254
```

此时外界就可以通过 SSH 服务远程连接到 Linux 系统了。

4.4.3 检查 DNS 解析文件是否设置正确

在 Linux 系统中,有两个文件用来指定系统到哪里寻找相关域名解析的库:分别是文件 /etc/host.conf 和 /etc/nsswitch.conf。/etc/host.conf 文件用于指定系统如何解析主机名,Linux 通过域名解析库来获得主机名对应的 IP 地址。下面是 CentOS 系统安装后默认的 /etc/host.conf 内容:

```
order hosts,bind
```

其中,order 指定主机名查询顺序,这里表示首先查找 /etc/hosts 文件对应的解析,如果没有找到对应的解析,接下来就根据 /etc/resolve.conf 指定的域名服务器进行解析。

/etc/nsswitch.conf 文件是由 SUN 公司开发的,用于管理系统中多个配置文件查询的

顺序。由于 `nsswitch.conf` 提供了更多的资源控制方式，因此它现在已经基本取代了 `hosts.conf`。虽然 Linux 系统中默认这两个文档都存在，但实际上起作用的是 `nsswitch.conf` 文件。

`nsswitch.conf` 文件每行的配置都以一个关键字开头，后跟冒号，紧接着是空白，然后是一系列方法的列表。

例如这段信息：

```
hosts:    files dns
```

表示系统首先查询 `/etc/hosts` 文件，如果没有找到对应的解析，就会去 DNS 配置文件指定的 DNS 服务器上解析。

清楚了 Linux 下域名解析的原理和过程，就可以根据这两个文件的设定，确定解析的顺序，从而判断域名解析可能出现的问题。

4.4.4 检查服务是否正常打开

在一个应用出现故障时，必须要检查的就是服务本身，比如服务是否开启，配置是否正确等。检查服务是否正确打开分为两步，第一步是查看服务的端口是否打开。

例如，我们不能用 `root` 用户 SSH 登录 10.10.80.89 这台 Linux 服务器，首先检查 `sshd` 服务的 22 端口是否打开：

```
[root@localhostinit.d]# telnet 10.10.80.89 22
SSH-2.0-OpenSSH_4.3
```

这个输出表示 10.10.80.89 的 22 端口对外开放，或者说 `sshd` 服务处于打开状态。如果没有任何输出，可能是服务没有启动，或者服务端口被屏蔽。

也可以在服务器上通过 `netstat` 命令检查 22 端口是否打开：

```
[root@localhostxinetd.d]# netstat -ntln
tcp        0      0 0.0.0.0:3306          0.0.0.0:*            LISTEN
tcp        0      0 :::80                :::*                  LISTEN
tcp        0      0 :::22                :::*                  LISTEN
```

可以看到，22 端口在服务器上打开的，同时，在服务器上打开的还有 3306、80 端口。

接着进行第二步检查，既然服务已经打开，就可能是 `sshd` 服务配置的问题，检查 `sshd` 服务端配置文件 `/etc/ssh/sshd_config`，发现有下面一行信息：

```
PermitRootLogin no
```

由此可知是 SSH 服务端配置文件限制了 `root` 用户不能登录系统，如果需要 `root` 用户登录系统，只需更改为如下即可：

```
PermitRootLogin yes
```


到这里为止，我们通过对端口和服务配置文件的层层检查，最终找到了问题的根源。需要说明的是，这里的重点不是讲述如何让 root 登录 Linux 系统，而是要通过这个例子让读者学会处理类似问题的思路和方法。

4.4.5 检查访问权限是否打开

1. 检查系统防火墙 iptables 的状态

当某些服务不能访问时，一定要检查是否被 Linux 本机防火墙 iptables 屏蔽了，可以通过“iptables -L”命令查看 iptables 的配置策略。例如不能访问某台 Linux 服务器提供的 WWW 服务，通过检查系统网络、域名解析都正常，并且服务也能正常启动，然后检查服务器的 iptables 策略配置，信息如下：

```
[root@localhost ~]# iptables -L -n
Chain INPUT (policy DROP)
targetprot opt source destination
Chain FORWARD (policy ACCEPT )
targetprot opt source destination
Chain OUTPUT (policy DROP )
targetprot opt source destination
```

从上面的输出可知，这台 Linux 服务器仅仅设置了预设策略，而致命的是将 INPUT 链和 OUTPUT 链都设置为 DROP，也就是所有外部数据不能进入服务器，服务器数据也不能出去，这样的设置相当于没有网络。

为了能访问这台服务器提供的 WWW 服务，增加两条策略即可：

```
[root@localhost ~]# iptables -A INPUT -i eth0 -p tcp --dport 80 -j ACCEPT
[root@localhost ~]# iptables -A OUTPUT -p tcp --sport 80 -m state --state
ESTABLISHED -j ACCEPT
```

这样一来，网络上的其他人就能够访问这台 Linux 服务器的 WWW 服务了。

2. 检查 SELinux 是否打开

SELinux 是个系统级的安全防护工具，它可以最大限度地保证 Linux 系统的安全，但是 SELinux 有时也会给 Linux 下软件的运行带来一些问题，这些问题大部分是由于对 SELinux 不了解造成的。为了迅速定位问题，最简单的方法是先关闭 SELinux，然后测试软件运行是否正常，这不是个好方法，但是对于判断问题往往是很有用的。SELinux 是个很好的安全访问控制软件，可是对于还不能熟练运用 SELinux 访问控制策略的读者，还是建议将它暂时关闭，等到对 Linux 有了更深入的认识后，再开启 SELinux 不失为一个明智的策略。

4.4.6 检查局域网主机之间联机是否正常

通过上面 5 步的检查，Linux 系统自身的问题已经基本排除，接下来需要扩展到 Linux 主机之外的网络环境，要检查网络之间的连通是否存在故障，可以先通过 ping 命令测试局域网主机之间的连通性，然后 ping 网关，检测主机到网关的通信是否正常。

例如下面这台服务器，在这台主机上 ping 网关，输出信息如下：

```
[root@localhost ~]# ping 10.10.80.1
PING 10.10.80.1 (10.10.80.1) 56(84) bytes of data.
64 bytes from 10.10.80.1: icmp_seq=1 ttl=64 time=2231 ms
64 bytes from 10.10.80.1: icmp_seq=2 ttl=64 time=2292 ms
64 bytes from 10.10.80.1: icmp_seq=3 ttl=64 time=2140 ms
64 bytes from 10.10.80.1: icmp_seq=4 ttl=64 time=1910 ms
```

很明显，这台主机到网关的延时很长，然后继续测试局域网中的其他主机到这台服务器的 ping 状态，延时也非常长，此时基本可以判断出这台服务器的网络连接存在问题，最后更换网线后，ping 延时恢复到正常状态了，只有 0.02 ms 左右。

至此，我们对排查网络故障的常见方法和思路进行了简单介绍，其实任何网络故障的出现都是有原因的，只要根据上面给出的解决问题思路逐一排查，99% 的问题都能得到很好解决。

本书介绍 Linux 系统运维过程中常见的一些故障和案例，这些案例都来自实际的客户环境，每个案例都会讲述一个小问题，这些问题非常简单，相信很多读者也都遇到过，但是需要说明的是，解决问题不是本章讲述的重点，系统运维过程中的错误也千差万别，不可能把每个问题都讲述一遍，这里重点讲述的是解决问题的思路，对每个问题的讲述，我们都是从错误现象开始介绍，然后是解决问题的思路，接着是问题的排查过程，最后给出解决问题的方法，通过这样由点到面的介绍，最终使得读者掌握解决问题的统一方法和思路，有了这样的思路，相信所有问题都能迎刃而解。

5.1.1 su 切换用户带来的疑惑

这是一个客户的案例，客户的一台 Oracle 数据库服务器突然宕机了，由于在线业务的需求，客户没有考虑太多就直接重启了服务器，系统重启后似乎没有出现问题，可是接下来，当客户准备切换到 oracle 用户下继续数据库时，怎么都无法进行 su 切换，于是问题出现了。

1. 案例现象

在 root 用户下，su 切换到普通用户 oracle 却发生了错误，如图 5-1 所示。

Linux 故障排查案例实战

5.1 常见系统故障案例

本章主要介绍 Linux 系统运维过程中常见的一些故障和案例，这些案例都来自实际的生产环境，每个案例都会讲述一个小问题，这些问题非常简单，相信很多读者也都遇到过。但是需要说明的是，解决问题不是本章讲述的重点，系统运维过程中的错误也千差万别，不可能把每个问题都讲述一遍，这里重点讲述的是解决问题的思路。对每个问题的讲述，我们都从错误现象开始介绍，然后是解决问题的思路。接着是问题排查过程，最后给出解决问题的方法。通过这样抽丝剥茧的介绍，最终使读者掌握解决问题的统一方法和思路，有了这样的思路，相信所有问题都能迎刃而解。

5.1.1 su 切换用户带来的疑惑

这是一个客户的案例，客户的一台 Oracle 数据库服务器突然宕机了，由于在线业务的需要，客户没有考虑太多就直接重启了服务器，系统重新启动后没有出现问题，可是接下来，当客户准备切换到 oracle 用户下启动数据库时，怎么都无法进行 su 切换，于是问题出现了。

1. 案例现象

在 root 用户下，su 切换到一个普通用户 oracle 却发生了错误，如图 5-1 所示。

```
[root@localhost ~]# su - oracle
su: warning: cannot change directory to /home/oracle: Permission denied
su: /bin/bash: Permission denied
```

图 5-1 su 切换发生 Permission denied 错误

于是, 尝试直接通过 oracle 用户登录系统, 发现此时的 oracle 用户也无法登录了, 出现与上面同样的错误。

2. 解决思路

从上面错误提示可知是权限出现了问题, 那么可以从权限入手进行排查, 基本思路如下:

- 用户目录 /home/oracle 权限问题。
- su 程序执行权限问题。
- 程序依赖的共享库权限问题。
- SELinux 问题导致。
- 系统根空间问题。

3. 排查问题

根据上面的思路, 逐一检查, 考虑到 su 在切换到 oracle 用户时会读取 oracle 目录下的环境变量配置文件, 因此, 首先检查 /home/oracle 目录的权限是否存在问题:

```
[root@localhost home]# ls -al /home|grep oracle
drwx----- 4 oracle oinstall 4096 01-31 10:45 oracle
```

从输出可知, /home/oracle 目录的属主是 oracle 用户, 而 oracle 用户对这个目录具有“rwX”权限, 因此, oracle 用户目录的权限设置是正确的, 可以排除这个问题。

接着检查 su 执行权限问题:

```
[root@localhost home]# ll /bin/su
-rwsr-xr-x 1 root root 24120 2007-11-30 /bin/su
```

可见 su 命令执行权限也没有问题, 这个问题也排除了。

继续检查 su 依赖的共享库权限, 使用 ldd 命令检查 su 命令依赖的共享库文件, 如图 5-2 所示。

根据上面的操作, 依次检查 su 命令依赖的每个库文件的权限, 发现也都是正常的, 都具有可执行权限, 因此, 共享库的问题也排除了。

根据上面的思路, 继续检查 SELinux 的设置, 执行命令如图 5-3 所示。

由输出可知, SELinux 处于关闭状态, 这个原因也排除了。


```

[root@localhost home]# ldd /bin/su
linux-gate.so.1 => (0x005d3000)
libpam.so.0 => /lib/libpam.so.0 (0x00605000)
libpam_misc.so.0 => /lib/libpam_misc.so.0 (0x005dd000)
libcrypt.so.1 => /lib/libcrypt.so.1 (0x006c8000)
libdl.so.2 => /lib/libdl.so.2 (0x00b66000)
libc.so.6 => /lib/libc.so.6 (0x00c94000)
libaudit.so.0 => /lib/libaudit.so.0 (0x00e37000)
/lib/ld-linux.so.2 (0x0065f000)
[root@localhost home]# ll /lib/ld-linux.so.2
lrwxrwxrwx 1 root root 9 2011-06-09 /lib/ld-linux.so.2 -> ld-2.5.so
[root@localhost home]# ll /lib/ld-2.5.so
-rwxr-xr-x 1 root root 129900 2011-04-27 /lib/ld-2.5.so

```

图 5-2 通过 ldd 命令检查 su 命令依赖的共享库文件

```

[root@localhost home]# grep "SELINUX" /etc/selinux/config
SELINUX=disabled
SELINUXTYPE=targeted

```

图 5-3 检查 SELinux 的设置

到目前为止，问题变得扑朔迷离，到底是哪里出现问题了呢？作为 Linux 运维人员，例行检查系统根分区状态是非常必要的，那么首先检查一下根分区的磁盘空间大小，发现剩余空间还有很多，排除空间问题。既然报的错误是权限有问题，那么只要以权限为线索，不偏离这个核心就没错，于是继续尝试检查 /home 目录下各个用户的权限，执行命令如图 5-4 所示。

```

[root@localhost home]# ls -al
总计 56
drwxr-xr-x 12 root root 4096 2012-03-30 .
lrw-rw-rw- 27 root root 4096 03-23 23:39 ..
drwx----- 3 mysql mysql 4096 2011-03-21 mysql
drwx----- 3 nagios nagios 4096 2011-03-26 nagios
drwx----- 4 oracle oinstall 4096 01-31 10:45 oracle

```

图 5-4 检查 /home 目录下各个用户的权限

从输出看每个用户的目录权限，都是“rwx-----”，即“700”，完全没有问题。仔细检查思路，发现当前的目光一直停留在用户对应的目录上，而忽略了其他输出信息，而问题就藏在之前没有关注的信息中。在这个命令输出的前两行中，第一行权限对应的目录是“.”，代表当前目录，也就是 /home 目录，权限为“rwxr-xr-x”，即“755”，第二行权限对应的目录是“..”，也就是根目录，权限却为“rw-rw-rw-”，即“666”，此时，问题终于查找到了，原来是根目录权限问题。

将根目录权限设置为“rw-rw-rw-”，显然是不正常的。在正常情况下根目录的权限应该是“755”，为何设置成了这样，很大的可能是误操作。通过 ls 命令查看根目录的权限时展示不是很清楚，也容易被很多运维人员忽略，其实我们可以通过另一个命令 stat 来详细查看每个目录的权限，如图 5-5 所示。

```

[root@localhost home]# stat /
  File: "/"
  Size: 4096      Blocks: 16      IO Block: 4096   目录
Device: 801h/2049d Inode: 2        Links: 27
Access: (0666/drw-rw-rw-)  Uid: (  0/   root)   Gid: (  0/   root)
Access: 2013-03-30 14:31:20.000000000 +0800
Modify: 2013-03-23 23:39:49.000000000 +0800
Change: 2013-03-30 14:31:17.000000000 +0800

```

图 5-5 通过 stat 命令查看根目录权限

通过这个命令，可以很清晰地看到，根目录的权限是“0666”，这才是导致 su 执行失败的根本原因。

4. 解决问题

知道了问题产生的原因，解决问题就非常简单，执行如下命令：

```
[root@localhost ~]# chmod 755 /
```

然后就可顺利执行 su 切换命令。

最后，做个简单的总结，这个问题主要是由于根目录没有可执行权限，而 Linux 下所有的操作都是在根目录下进行的，进而导致 /home/oracle 目录没有执行权限。其实根目录权限的丢失对于系统中运行的每个用户存在同样的影响。因此，在权限出现问题时，一定要注意根目录的权限。

5.1.2 “Read-only file system” 错误与解决方法

这个问题经常发生在有大量磁盘读写操作且磁盘分区很大的环境中，下面简单描述下此案例的应用环境。这是一个 Web 服务器故障案例，客户利用两台服务器加一个磁盘阵列做了一个双机热备的 Web 系统，所有网站数据都存储在磁盘阵列中，两台服务器共享一个磁盘阵列分区，在正常情况下主机挂载磁盘阵列，分区提供网站服务，主机故障时备机接管磁盘阵列分区继续提供网站服务。

1. 案例现象

接到客户电话说他们的网站无法添加数据了，不过网站还可以正常访问，服务器和磁盘阵列也没有任何告警信息。

2. 解决思路

根据这个简单信息，基本排查思路如下：

- ☐ 网站程序可能出现问题了。
- ☐ 服务器磁盘故障。

3. 排查问题

首先通知研发人员对网站程序问题进行排查。经过检查,并没有发现程序有问题,而在程序日志中发现了一条信息:

```
java.lang.RuntimeException: Cannot make directory: file:/www/data/html/2013-03-30
```

根据这个输出可知,程序不能创建目录,那么尝试手动创建一个目录,登录 Web 服务器,在 /www/data/html 目录下创建一个目录 test,操作如下:

```
[root@localhost html]# mkdir test
mkdir: cannot create directory `test': Read-only file system
```

从这个输出信息可知, /www/data/html 目录所在的磁盘分区出现了问题,通过检查发现, /www/data/html 目录正是挂载的磁盘阵列分区,于是问题原因找到了。

4. 解决问题

磁盘出现“Read-only file system”的原因有很多种,可能是文件系统数据块出现不一致导致的,也有可能是磁盘故障造成的。主流的 ext3、ext4 文件系统都有很强的自我修复机制,对于简单的错误,文件系统一般可自行修复,当遇到致命错误无法修复时,文件系统为了保证数据一致性和安全,会暂时屏蔽文件系统的写操作,将文件系统变为只读,进而出现了上面的“Read-only file system”现象。

手工修复文件系统错误的命令是 fsck,在修复文件系统前,最好卸载文件系统所在的磁盘分区:

```
[root@localhost ~]# umount /www/data
umount: /www/data: device is busy
```

提示无法卸载,可能这个磁盘中还有文件对应的进程在运行,检查如下:

```
[root@localhost ~]# fuser -m /dev/sdb1
/dev/sdb1:          8800
```

接着检查一下 8800 端口对应的是什么进程,如图 5-6 所示。

```
[root@localhost ~]# ps -ef|grep 8800
root      1266   1230    0 17:46 pts/0    00:00:00 grep 8800
root      8800     1    0 2012 ?        00:00:28 /usr/local/apache2/bin/httpd -k start
nobody    8801   8800    0 2012 ?        00:00:00 /usr/local/apache2/bin/httpd -k start
nobody    8802   8800    0 2012 ?        00:00:00 /usr/local/apache2/bin/httpd -k start
nobody    8803   8800    0 2012 ?        00:00:00 /usr/local/apache2/bin/httpd -k start
nobody    8808   8800    0 2012 ?        00:00:00 /usr/local/apache2/bin/httpd -k start
nobody    9106   8800    0 2012 ?        00:00:00 /usr/local/apache2/bin/httpd -k start
nobody    27800   8800    0 Mar28 ?       00:00:00 /usr/local/apache2/bin/httpd -k start
```

图 5-6 查询 8800 端口对应的进程信息

原来是系统的 apache 进程还没有停止，停止 apache，成功卸载磁盘，操作如下：

```
[root@localhost ~]# /usr/local/apache2/bin/apachectl stop
[root@localhost ~]# umount /www/data
```

最后，执行修复操作，如图 5-7 所示。

```
[root@localhost ~]# fsck -V -a /dev/sdb1
fsck 1.39 (29-May-2006)
[/sbin/fsck.ext3 (1) -- /www/data] fsck.ext3 -a /dev/sdb1
.....
/www/data: recovering journal
/www/data: clean, 11980860/122060800 files, 87710849/488241447 blocks
```

图 5-7 用 fsck 命令修复磁盘分区

修复过程比较简单，上面省略了很多输出信息。修复的时间根据磁盘大小和文件系统损坏程度而定。如果有些数据无法修复，会提示是否删除，此时可根据情况选择。修复完成后，被删除的文件会保留在对应磁盘分区挂载点的 lost+found 目录中。

修复完成后，执行挂载操作：

```
[root@localhost ~]# mount /dev/sdb1 /www/data
```

最后，在 /www/data 目录下验证是否可以成功创建文件，至此，问题圆满解决。

5.1.3 “Argument list too long” 错误与解决方法

作为一名运维人员，对这个错误并不陌生，在执行 rm、cp、mv 等命令时，如果要操作的文件数很多，可能会使用通配符批量处理大量文件，这时就可能会出现“Argument list too long”这个问题。

1. 错误现象

这是一台 MySQL 数据库服务器，在系统中运行了很多定时任务，某天通过如下 crontab 命令又添加了一个计划任务，退出时系统报错。

```
#crontab -e
```

编辑完成后，保存退出，就出现如图 5-8 所示的错误。

```
crontab: installing new crontab
cron/tmp/crontab.XXXXHkEy3L: No space left on device
crontab: edits left in /tmp/crontab.XXXXHkEy3L
```

图 5-8 crontab 命令编辑完成后发生的错误

2. 解决思路

根据上面的报错信息，基本判定是磁盘空间满了，那么首先从检查服务器的磁盘空间开

始,先检查 /tmp 磁盘空间,然后检查根分区的磁盘空间,最后检查系统其他分区的磁盘空间。

3. 问题排查

通过 df 命令查看了这个服务器上所有磁盘分区的情况, /tmp 分区空间还有很多,根分区也还有很大剩余空间,都不存在问题,最后发现是 /var 磁盘分区空间已经达到 100% 了。至此已经定位了问题,是 /var 磁盘空间爆满导致的,因为 crontab 会在保存时将文件信息写到 /var 目录下,然而这个磁盘没有空间了,所以报错也是理所当然了。

4. 解决问题

接着通过“du -sh”命令检查 /var 目录下所有文件或目录的大小,发现 /var/spool/clientmqueue 目录占用了 /var 整个分区大小的 90%,那么 /var/spool/clientmqueue 目录下的文件都是怎么产生的呢,能否删除?下面简单介绍下 /var/spool/clientmqueue 目录的文件是怎么生成的。

打开 /var/spool/clientmqueue 目录下的一些文件,发现都是一些邮件信息,邮件内容大多是关于 Cron Daemon 的,其实 /var/spool/clientmqueue 就是一个邮件暂存的目录。Linux 服务器在默认情况下会发一些邮件,比如当 cron 执行的程序有输出内容时,就会发送邮件信息到执行 cron 进程的用户。在发送邮件时,系统首先会把邮件复制到 /var/spool/clientmqueue 目录下,然后等待 MTA(Mail Transfer Agent) 程序来处理。而 MTA 主要的功能是把这个目录中的邮件转移到 /var/spool/mqueue 目录下,然后再通过 sendmail 服务发送到真正的目的地。检查这个服务器的 sendmail 服务,发现其没有开启,这样 /var/spool/clientmqueue 目录非常大的原因就找到了:没有发送邮件的客户端服务,所有邮件就都堆积在这个目录下了。

在确认完这些邮件内容都没用后,切换到 /var/spool/clientmqueue 目录下,执行 rm 命令删除所有的文件时,出现如下错误:

```
[root@localhost clientmqueue]# rm *  
/bin/rm: argument list too long
```

此时出现了本节开头谈到的问题。

当在 Linux 系统中试图传递太多参数给一个系统命令时,就会出现“Argument list too long”错误。这是 Linux 系统一直以来都有的限制。查看这个限制可以通过命令“getconf ARG_MAX”来实现,如图 5-9 所示。

这是 CentOS 6.x 版本的一个最大值,而在 CentOS 5.x 中,这个值相对较小,如图 5-10 所示。

所以这个问题更多的是发生在 Linux 低版本中。

```
[root@localhost ~]# getconf ARG_MAX
2621440
[root@localhost ~]# more /etc/issue
CentOS release 6.3 (Final)
Kernel \r on an \m
```

图 5-9 查看 CentOS 6.3 版本的 Linux 系统传递参数限制

```
[root@localhost ~]# getconf ARG_MAX
131072
[root@localhost ~]# more /etc/issue
CentOS release 5.8 (Final)
Kernel \r on an \m
```

图 5-10 查看 CentOS 5.8 版本的 Linux 系统传递参数限制

知道了产生问题的原因，解决方法就很多了，这里提供四种解决此问题的方法，下面分别进行介绍。

(1) 手动把命令行参数分成较小的部分

例如：

```
rm [a-n]* -rf
rm [o-z]* -rf
```

这种方法最简单，但是相对较低级，因为必须知道怎么平均分割文件，同时对于文件数目极多的情况，需要输入很多次命令。

(2) 使用 find 命令删除

基本原理是通过 find 命令筛选文件列表，把符合要求的文件传递给一系列命令。这种方法是最简洁的，也是最有效的。

例如：

```
find /var/spool/clientmqueue -type f -print -exec rm -f {} \;
```

但是这种方法也有缺点，需要遍历所有文件，因而在文件数量极多时比较耗时。

(3) 通过 shell 脚本

这种方法是通过编写一个 shell 脚本，然后通过循序语句实现，与 find 方法类似。

例如，可以编写如下脚本：

```
#!/bin/bash

# 设定需要删除的文件夹
RM_DIR='/var/spool/clientmqueue'

cd $RM_DIR
for I in `ls`
```

```
do
rm -f $I
done
```

(4) 重新编译 Linux 内核

这种方法需要手动增加内核中分配给命令行参数的页数，打开 kernel source 下面的 include/linux/binfmts.h 文件，找到如下行：

```
# define MAX_ARG_PAGES 32
```

将“32”改为更大的值，例如 64 或 128，然后重新编译内核。

此种方法永久有效，可以彻底解决问题，但是比较复杂，推荐给高级用户使用，没有 Linux 经验的用户不建议用这种方法。

5.1.4 inode 耗尽导致应用故障

1. 错误现象

客户的一台 Oracle 数据库服务器在关机重启后，Oracle 监听无法启动，提示错误如图 5-11 所示。

```
TNS-12549: TNSoperating system resource quota exceeded
TNS-12560: TNSrotocol adapter error
TNS-00519: Operating system resource quota exceeded
Linux Error: 28: No space left on device
```

图 5-11 inode 耗尽故障现象

从输出信息判断，应该是磁盘空间耗尽导致 Oracle 监听无法启动，因为 Oracle 在启动监听时需要创建监听日志文件，而上面三个 TNS 错误产生的原因都是由最后一行错误导致的，于是首先检查系统磁盘空间，如图 5-12 所示。

```
[www@localhost ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda6       20G   4.4G   15G   24% /
/dev/sda8       1.2T  321G   828G   28% /backup
/dev/sda5       20G   3.6G   15G   20% /usr
/dev/sda2       99G   80G   14G   80% /home
/dev/sda3       20G   16G   3.2G   84% /var
/dev/sda1       99M   18M   76M   20% /boot
tmpfs           16G     0    16G    0% /dev/shm
```

图 5-12 查看故障服务器磁盘空间信息

从磁盘输出信息可知，所有分区磁盘空间都还有不少剩余，而 Oracle 监听写日志的路径在 /var 分区下，虽然 /var 分区只剩下 3.2GB 可用磁盘空间，但是这对于写一个监听日志

文件来说足够了，为什么还提示空间不足呢？

2. 解决思路

既然错误提示与磁盘空间有关，那就深入研究下关于磁盘空间的问题，在 Linux 系统对磁盘空间的占用分为三个部分：第一个是物理磁盘空间，第二个是 inode 节点所占用的磁盘空间，第三个是 Linux 用来存放信号量的空间，而平时接触较多的是物理磁盘空间，对第二个和第三个空间的问题接触较少。既然不是物理磁盘空间的问题，接着检查是否是 inode 节点耗尽的问题，通过执行“df -i”查看系统可用的 inode 节点，如图 5-13 所示。

Filesystem	Inodes	IUsed	IFree	IUse%	Mounted on
/dev/sda6	5244736	131285	5113451	3%	/
/dev/sda8	160989184	7387	160981797	1%	/backup
/dev/sda5	5244736	156624	5088112	3%	/usr
/dev/sda2	13107200	333727	12773473	3%	/home
/dev/sda3	5244736	5244736	0	100%	/var
/dev/sda1	26104	41	26063	1%	/boot
tmpfs	4110598	1	4110597	1%	/dev/shm

图 5-13 查看磁盘分区的 inode 使用信息

由输出可知，果然是 inode 节点耗尽导致无法写日志文件。由于 inode 被全部用完了，虽然还有可用磁盘空间，但是文件系统已经无法再记录这些空余空间了，因此也就不能再创建新文件或文件夹了。由于涉及了 inode 知识，接下来就简单介绍下 Linux 中 inode 的概念。

在 Linux 系统中，文件由数据块和元数据组成，数据块就是多个连续性的扇区，是文件存取的最小单位。块（block）的大小，最常见的是 4KB，即连续 8 个 sector 组成一个 block。而元数据用来记录文件的创建者、文件的创建日期、文件的大小等，这种存储文件元数据信息的区域就叫做 inode，或者称为“索引节点”。

由于 inode 也是用来存储文件相关属性信息的，因此 inode 也会消耗硬盘空间，在硬盘格式化的时候，操作系统会自动将硬盘分成两个区域。一个是数据区，存放文件数据；另一个是 inode 区（inode table），存放 inode 所包含的信息。

每个 inode 节点的大小一般是 128B 或 256B。inode 节点的总数在格式化文件系统的时候，就已经确定，可以通过如下命令查看某个磁盘分区 inode 的总数：

```
[root@localhost ~]# dumpe2fs -h /dev/sda3|grep 'Inode count'
dumpe2fs 1.39 (29-May-2006)
Inode count: 5244736
```

举个形象的例子，如果将文件系统比作一本书，那么，inode 就是这本书的目录。在格式化文件系统的时候，这本书的最大目录数已经确定了。在写书（保存文件到磁盘）的过程

中,可能发生这样的情况:纸用完了(磁盘空间不足),此时肯定无法保存新的文件;但是还存在另外一种情况,就是目录写完了(inode 节点全部分配完了),在这种情况下,虽然还有纸(磁盘空间),但是目录(inode)已经没有了,在文件系统上当然不能新建文件了,因为没有了目录,就无法通过索引找到文件。

另外,每个 inode 都有一个号码,操作系统用 inode 号码来区分不同的文件。通过“ls -i”命令,可以查看文件名对应的 inode 号,例如:

```
[root@localhost ~]# ls -i install.log
325762 install.log
```

如果要查看这个文件更详细的 inode 信息,可以通过 stat 命令实现,如图 5-14 所示。

```
[root@localhost ~]# stat install.log
  File: 'install.log'
  Size: 39224          Blocks: 88          IO Block: 4096   regular file
Device: 806h/2054d    Inode: 325762       Links: 1
Access: (0644/-rw-r--r--)  Uid: (  0/   root)   Gid: (  0/   root)
Access: 2011-07-26 19:09:02.000000000 +0800
Modify: 2011-07-26 19:17:38.000000000 +0800
Change: 2011-07-26 19:17:43.000000000 +0800
```

图 5-14 查看某文件 inode 的详细信息

3. 解决问题

知道这个故障是由 inode 导致的后,接下来就要查看 /var 目录下为何耗尽了 inode,通过检查发现 /var/spool/clientmqueue/ 这个目录里面的文件仅 500 多万个,至于产生的原因,分析后确定应该是系统的 crontab 导致的,因为系统开了多个 crontab 任务,而如果 crontab 任务没有重定向,默认就会在这个目录下创建一个文件,日积月累,此目录下的小文件就会超级多。解决的方法很简单,删除这些没用的文件即可。删除的方法是直接使用 rm 命令,这时肯定会提示“Argument list too long”的错误,而解决这个的方法前面章节已经有过详细介绍,通过如下命令即可完成:

```
[root@localhost ~]# find /var/spool/clientmqueue/ -name "*" -exec rm -rf {} \;
```

删除日志文件后,再次启动 Oracle 监听,可以顺利实现启动,查看发现新的监听日志文件已经生成,至此,问题得到圆满解决。

5.1.5 文件已删除但空间不释放的原因

1. 错误现象

运维的监控系统发来通知,报告一台服务器空间满了,登录服务器查看,根分区确实

没有空间了，如图 5-15 所示。

```
[root@localhost ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda3        97G   97G    0 100% /
/dev/sda1        99M   18M   76M  20% /boot
tmpfs            16G    0   16G   0% /dev/shm
```

图 5-15 查看服务器磁盘空间

这里首先说明一下服务器的一些删除策略，由于 Linux 没有回收站功能，所以线上服务器上所有要删除的文件都会先移动到系统 /tmp 目录下，然后定期清除 /tmp 目录下的数据。这个策略本身没有问题，但是通过检查发现这台服务器的系统分区中并没有单独划分 /tmp 分区，这样 /tmp 下的数据其实占用了根分区的空间。既然找到了问题，那么删除 /tmp 目录下一些占空间较大的数据文件即可，检查 /tmp 下最大的三个数据文件，如图 5-16 所示。

```
[root@localhost ~]# du -sh /tmp/*|sort -nr|head -3
66G    /tmp/access_log
36K    /tmp/hsperfdata_root
36K    /tmp/hsperfdata_mapred
```

图 5-16 查看 /tmp 下最大的前三个数据文件

通过命令输出发现在 /tmp 目录下有个 66GB 大小的文件 access_log，这个文件应该是 Apache 产生的访问日志文件，从日志大小来看，应该是很久没有清理 Apache 日志文件了，基本判定是这个文件导致的根空间爆满，在确认此文件可以删除后，执行如下删除操作：

```
[root@localhost ~]# rm /tmp/access_log
```

接着查看系统根分区空间是否释放，如图 5-17 所示。

```
[root@localhost ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda3        97G   97G    0 100% /
/dev/sda1        99M   18M   76M  20% /boot
tmpfs            16G    0   16G   0% /dev/shm
```

图 5-17 查看磁盘空间是否释放

从输出可以看到，根分区空间仍然没有释放，这是怎么回事？

2. 解决思路

一般来说不会出现删除文件后空间不释放的情况，但是也存在例外，比如文件被进程锁定，或者有进程一直在向这个文件写数据等，要理解这个问题，就需要知道 Linux 下文件的存储机制和存储结构。

一个文件在文件系统中的存放分为两个部分：数据部分和指针部分，指针位于文件系统的 meta-data 中，在将数据删除后，这个指针就从 meta-data 中清除了，而数据部分存储在磁盘中。在将数据对应的指针从 meta-data 中清除后，文件数据部分占用的空间就可以被覆盖并写入新的内容，之所以在出现删除 access_log 文件后，空间还没释放，就是因为 httpd 进程还在一直向这个文件写入内容，导致虽然删除了 access_log 文件，但是由于进程锁定，文件对应的指针部分并未从 meta-data 中清除，而由于指针并未删除，系统内核就认为文件并未删除，因此通过 df 命令查询空间并未释放也就不足为奇了。

3. 问题排查

既然有了解决问题的思路，那么接下来看看是否有进程一直在向 access_log 文件中写数据，这里需要用到 Linux 下的 lsof 命令，通过这个命令可以获取一个仍然被应用程序占用的已删除文件列表，命令执行如图 5-18 所示。

```
[root@localhost ~]# lsof | grep delete
mysqld 3362 mysql 4u REG 253,0 0 75694090 /tmp/ibr0ZPnv (deleted)
mysqld 3362 mysql 5u REG 253,0 0 75694091 /tmp/ibcPLKkn (deleted)
mysqld 3362 mysql 6u REG 253,0 0 75694092 /tmp/ib8nXFhf (deleted)
mysqld 3362 mysql 7u REG 253,0 0 75694093 /tmp/ibU3bHf7 (deleted)
mysqld 3362 mysql 11u REG 253,0 0 75694095 /tmp/ibvfslp2 (deleted)
httpd 2986 apache 3w REG 8 70866960384 75694218 /tmp/access_log(deleted)
```

图 5-18 查看被应用程序锁定的已删除文件列表

从输出结果可以看到，/tmp/access_log 文件被进程 httpd 锁定，而 httpd 进程还一直向这个文件写入日志数据。从第 7 列可知，这个日志文件大小约 70GB，而系统根分区总大小才 100GB，由此可知，这个文件就是导致系统根分区空间耗尽的罪魁祸首。最后一列的“deleted”状态说明这个日志文件已经被删除，但由于进程还在一直向此文件写入数据，因此空间并未释放。

4. 解决问题

到这里问题就基本排查清楚了，解决这一类问题的方法有很多种，最简单的方法是关闭或重启 httpd 进程，当然也可以重启操作系统，不过这些并不是最好的方法。对待这种进程不停对文件写日志的操作，要释放文件占用的磁盘空间，最好的方法是在线清空这个文件，具体可以通过如下命令完成：

```
[root@localhost ~]# echo " " >/tmp/access_log
```

通过这种方法，磁盘空间不但可以马上释放，也可保障进程继续向文件写入日志，这种方法经常用于在线清理 Apache、Tomcat、Nginx 等 Web 服务产生的日志文件。

5.1.6 “Too many open files” 错误与解决方法

1. 问题现象

这是一个基于 Java 的 Web 应用系统，在后台添加数据时提示无法添加，于是登录服务器查看 Tomcat 日志，发现如下异常信息：

```
java.io.IOException: Too many open files
```

通过这个报错信息，基本判断是系统可用的文件描述符不够了，由于 Tomcat 服务是系统 www 用户启动的，于是以 www 用户登录系统，通过“ulimit -n”命令查看系统可以打开最大文件描述符的数量，输出如下：

```
[www@tomcatserver ~]$ ulimit -n
65535
```

可以看到这台服务器设置的最大可打开的文件描述符已经是 65535 了，这么大的值应该够用了，但是为什么提示这样的错误呢？

2. 解决思路

这个案例涉及 Linux 下 ulimit 命令的使用，这里简单介绍下 ulimit 的作用和使用技巧。ulimit 主要用来限制进程对资源的使用情况，它支持各种类型的限制，常用的有：

- ☐ 内核文件的大小限制。
- ☐ 进程数据块的大小限制。
- ☐ shell 进程创建文件大小限制。
- ☐ 可加锁内存大小限制。
- ☐ 常驻内存集的大小限制。
- ☐ 打开文件句柄数限制。
- ☐ 分配堆栈的最大大小限制。
- ☐ CPU 占用时间限制用户最大可用的进程数限制。
- ☐ shell 进程所能使用的最大虚拟内存限制。

ulimit 使用的基本格式为：

```
ulimit [options] [limit]
```

具体的 ulimit 参数（即 options）含义如表 5-1 所示。

表 5-1 ulimit 参数的含义

参数	含义
-a	显示当前系统所有的 limit 资源信息
-H	设置硬资源限制，一旦设置不能增加
-S	设置软资源限制，设置后可以增加，但是不能超过硬资源设置
-c	最大的 core 文件的大小，以 blocks 为单位
-f	进程可以创建文件的最大值，以 blocks 为单位
-d	进程最大的数据段的大小，以 Kbytes 为单位
-m	最大内存大小，以 Kbytes 为单位
-n	可以打开的最大文件描述符的数量
-s	线程栈大小，以 Kbytes 为单位
-p	管道缓冲区的大小，以 Kbytes 为单位
-u	用户最大可用的进程数
-v	进程最大可用的虚拟内存，以 Kbytes 为单位
-t	最大 CPU 占用时间，以 s 为单位
-l	最大可加锁内存大小，以 Kbytes 为单位

在使用 ulimit 时，有以下几种使用方法。

(1) 在用户环境变量中加入

如果用户使用的是 bash，那么可以在用户目录的环境变量文件 .bashrc 或 .bash_profile 中加入“ulimit -u 128”来限制用户最多可以使用 128 个进程。

(2) 在应用程序的启动脚本中加入

如果应用程序是 Tomcat，那么可以在 Tomcat 的启动脚本 startup.sh 中加入“ulimit -n 65535”来限制用户最多可以使用 65535 个文件描述符。

(3) 直接在 shell 命令终端执行 ulimit 命令

这种方法的资源限制仅仅在执行命令的终端生效，在退出或关闭终端后，设置失效，并且这个设置不影响其他 shell 终端。

有时候为了方便起见，也可以将用户资源的限制统一由一个文件来配置，这个文件就是 /etc/security/limits.conf，该文件不但能对指定用户的资源进行限制，还能对指定组的资源进行限制。该文件的使用规则如下：

```
<domain> <type> <item> <value>
```

其中：

- domain 表示用户或用户组的名字，还可以使用“*”作为通配符，表示任何用户或用户组。
- type 表示限制的类型，可以有两个值：soft 和 hard，分别表示软、硬资源限制。
- item 表示需要限定的资源名称，常用的有 nofile、cpu、stack 等。分别表示最大打开句柄数、占用的 CPU 时间、最大的堆栈大小。

□ value 表示限制各种资源的具体数值。

除了 limits.conf 文件之外, 还有一个 /etc/security/limits.d 目录, 可以将资源限制创建一个文件放到这个目录中, 系统默认会首先读取这个目录下的所有文件, 然后才去读取 limits.conf 文件。在所有资源限制设置完成后, 退出 shell 终端, 再次登录 shell 终端后, ulimit 设置即可自动生效。

3. 解决问题

在了解 ulimit 知识后, 接着上面的案例, 既然 ulimit 设置没问题, 那么一定是设置没有生效导致的。接下来检查下启动 Tomcat 的 www 用户环境变量下是否添加了 ulimit 限制, 检查后发现, www 用户下并无 ulimit 资源限制。于是继续检查 Tomcat 启动脚本 startup.sh 文件是否添加了 ulimit 限制, 检查后发现也没有添加。最后考虑是否将限制加到了 limits.conf 文件中, 于是检查 limits.conf 文件, 操作如下:

```
[root@tomcatserver ~]# cat /etc/security/limits.conf|grep www
www soft nofile 65535
www hard nofile 65535
```

从输出可知, ulimit 限制加在了 limits.conf 文件中, 既然限制已经加了, 配置也没有错, 为何还会报错呢? 经过长时间思考, 判断只有一种可能, 那就是 Tomcat 的启动时间早于 ulimit 资源限制的添加时间, 于是首先查看下 Tomcat 的启动时间, 操作如下:

```
[root@tomcatserver ~]# more /etc/issue
CentOS release 6.3 (Final)
Kernel \r on an \m
[root@tomcatserver ~]# uptime
15:10:19 up 283 days, 5:37, 4 users, load average: 1.20, 1.41, 1.35
[root@tomcatserver ~]# pgrep -f tomcat
4667
[root@tomcatserver ~]# ps -eo pid,lstart,etime|grep 4667
4667 Sat Jul 6 09:33:39 2013 77-05:26:02
```

从输出可以看出, 这台服务器已经有 283 天没有重启过了, 而 Tomcat 是在 2013 年 7 月 6 日 9 点多启动的, 启动了近 77 天零 5 个半小时了, 接着继续看看 limits.conf 文件的修改时间, 操作如图 5-19 所示。

```
[root@tomcatserver ~]# stat /etc/security/limits.conf
File: '/etc/security/limits.conf'
Size: 1880      Blocks: 8      IO Block: 4096   regular file
Device: 802h/2050d    Inode: 36538264  Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/   root)   Gid: (    0/   root)
Access: 2013-07-12 19:27:10.000000000 +0800
Modify: 2013-07-12 15:48:49.000000000 +0800
Change: 2013-07-12 15:48:49.000000000 +0800
```

图 5-19 查看 limits.conf 文件的最后修改时间

通过 stat 命令可以很清楚地看出, limits.conf 文件最后的修改时间是 2013-07-12, 通过查询相关的 Linux 系统管理人员, 基本确认就是在这个时候添加的 ulimit 资源限制, 这样此案例的问题就很明确了。由于 ulimit 限制的添加时间晚于 Tomcat 最后一次的启动时间, 而在此期间内, Tomcat 服务一直未重启过, 操作系统也一直未重启过, 那么 ulimit 资源限制对于 Tomcat 来说始终是不生效的, 同时, 由于此操作系统是 CentOS 6.3, 系统默认的最大可用句柄数是 1024, java 进程采用的是 Linux 默认的这个值, 因此出现 “Too many open files” 的错误也是合乎情理的。

清楚问题之后, 解决问题的方法非常简单, 重启 Tomcat 服务即可。

5.2 Apache 常见错误故障案例

本节主要介绍 Linux 系统下 Web 运维过程中常见的一些故障和案例, 这些案例都是基于 Apache 的错误或者故障的, 虽然和系统并没有直接关系, 但是通过分析和排查发现, 其实大部分故障都与系统参数设置有很大关系, 所以, 下面的案例都基于 Linux 系统的故障进行分析和排查, 在介绍过程中, 除给出了故障的解决方法外, 同时也介绍了与故障相关的 Linux 系统知识, 而这些知识是判断和解决问题必不可少的。

5.2.1 “No space left on device” 错误与解决方法

1. 错误现象

这也是一个客户案例, 客户反映在执行 “apachectl start” 启动 Apache 时无报错信息, 但是还是不能访问网页。客户的网站是基于 Apache+PHP+MySQL 的在线交易平台, 听到客户描述的现象后, 第一反应是防火墙屏蔽了 HTTP 端口或 SELinux 的问题, 于是登录服务器查看相关信息, 如图 5-20 所示。

```
[root@localhost ~]# more /etc/issue
CentOS release 5.5 (Final)
Kernel \r on an \m
[root@localhost ~]# iptables -L
Chain INPUT (policy ACCEPT)
target    prot opt source               destination

Chain FORWARD (policy ACCEPT)
target    prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target    prot opt source               destination
[root@localhost ~]# more /etc/selinux/config | grep -v "#"|grep SELINUX
SELINUX=disabled
SELINUXTYPE=targeted
```

图 5-20 查看系统防火墙和 SELinux 状态

从输出可知, 防火墙所有策略都处于开放状态, 并未作任何限制, 而 SELinux 也处于关闭状态, 应该不是防火墙问题导致的。

既然不是防火墙拦截的问题, 那么看看 httpd 进程是否存在及 httpd 端口是否正常启动, 操作过程如图 5-21 所示。

```
[root@localhost ~]# ps -ef|grep httpd|grep -v "grep"|wc -l
0
[root@localhost ~]# netstat -antl|grep 80
[root@localhost ~]# /usr/local/apache2/bin/apachectl start
[root@localhost ~]# ps -ef|grep httpd|grep -v "grep"|wc -l
0
```

图 5-21 查看 httpd 进程运行状态

这个操作首先查看了服务器上的 httpd 进程, 发现并没有 httpd 进程运行, 同时 httpd 对应的端口 80 也并没有启动, 于是重新启动 Apache, 在启动 Apache 的过程中并没有报错, 启动完成后发现仍然 httpd 进程没有运行, 由此看来, 应该是 Apache 内部出现了问题。

2. 解决思路

在判定是 Apache 的问题后, 首先要看的就是 Apache 的启动日志, 在查看 Apache 的 error 日志后, 发现了一个可疑输出, 内容为:

```
No space left on device: mod_rewrite: could not create rewrite_log_lock Configuration Failed
```

看到这个错误提示, 感觉应该是磁盘空间耗尽导致的, 于是赶紧查看系统所有磁盘分区, 结果发现所有磁盘分区都还有很多可用空间, 这就有些奇怪了。

在前面的案例介绍中, 详细介绍了 Linux 下对磁盘空间的占用分为三个部分: 物理磁盘、inode 节点磁盘空间和信号量磁盘空间。通过检查服务器的物理磁盘空间, 发现仍有很多剩余, 因此排除物理磁盘空间的问题。接着通过 “df -i” 命令查看系统可用的 inode 节点, 发现每个分区可用的 inode 还有很多, 这样 inode 节点问题也被排除了, 那么应该是信号量磁盘空间耗尽导致的。

这里简单介绍下 Linux 信号量相关的知识。信号量是一种锁机制, 用于协调进程之间互斥的访问临界资源, 以确保某种共享资源不被多个进程同时访问。Linux 系统的信号量是用于进程间通信的。它有两种标准实现, 分别是 POSIX 及 System v, 现在大多数 Linux 系统都实现了这两种标准。这两种标准都可用于进行线程间的通信, 只是系统调用方式略有不同。

❑ System v 信号量通过系统调用 semget 来创建, 通过 Linux 命令 ipcs 即可显示进程间通信用的 System v 类型信号量及共享内存。

❑ POSIX 信号量可用于线程和进程间通信, 并可分为有名和无名两种。也可简单理解

为是否保存在磁盘上。有名的信号量会以文件形式保存在 /dev/shm 下，因此可用于不相关的进程间通信，而无名信号量只能用于线程间和父子进程间通信。

在对信号量有了简单了解后，可以发现 Apache 使用的进程间通信方式应该是 System v，因此就可以通过 ipcs 命令查看和解决这个问题了。

3. 解决问题

在解决这个问题之前，首先查看下 Linux 系统默认信号量的设置值是多少，执行如下命令：

```
[root@localhost ~]# cat /proc/sys/kernel/sem
250 32000 32 128
```

这 4 个输出值的含义如下：

□ SEMMSL，此参数用于控制每个信号集的最大信号数。

□ SEMMNS，此参数用于控制整个 Linux 系统中信号（而不是信号集）的最大数量。

□ SEMOPM，此参数用于控制每个 semop 系统调用可以执行的信号操作数。

□ SEMMNI，此内核参数用于控制整个 Linux 系统中信号集的最大数量。

接着通过 ipcs 命令查看 httpd 进程占用了多少信号量，执行如下命令：

```
[root@localhost ~]# ipcs -s | grep daemon
```

其中 daemon 是启动 Apache 进程的用户，默认是 daemon，也可能是 nobody 用户，根据实际环境而定。在执行完此命令后，发现很多基于 daemon 的信号量输出，问题终于找到了。解决信号量耗尽的方法很简单，通过 ipcrm 命令清除即可，最简单方法是执行如下命令组合：

```
[root@localhost ~]# ipcs -s | grep nobody | perl -e 'while (<STDIN>) { @a=split(/\s+/); print "ipcrmsem $a[1]`"}'
```

执行完上面这个命令后，再次启动 Apache，然后查看 httpd 进程是否启动，可以看到，此时 httpd 进程启动了，那么 Apache 也工作正常了。

5.2.2 apache(20014) 故障与解决方法

1. 错误现象

这是一个客户的简单案例。客户告知网站无法访问了，并且 Apache 也无法启动。客户的网站是基于 Apache+Tomcat+MySQL 构建的，于是登录服务器查看信息如图 5-22 所示。

```

Linux:~ # uname -a
Linux linux 2.6.5-7.97-default #1 Fri Jul 2 14:21:59 UTC 2004 i686 i686 i386 GNU/Linux
linux:~ # more /etc/issue
Welcome to SUSE LINUX Enterprise Server 9 (i586) - Kernel \r (\l).
linux:~ # /usr/local/apache2/bin/apachectl start
(20014)Internal error: Error retrieving pid file logs/httpd.pid

```

图 5-22 Apache (20014)Internal error 现象

这里提示的是 httpd.pid 文件的错误，熟悉 Apache 的读者应该知道 httpd.pid 文件其实是 Apache 的进程 pid 文件，Apache 的启动进程 ID 就放在这个文件中。

2. 解决思路

既然提示 httpd.pid 这个文件有问题，那么就先看看这个文件是否存在，以及其中的内容是什么。查看 httpd.pid 文件，操作如下：

```
linux:~ # more /usr/local/apache2/logs/httpd.pid
```

发现这个文件存在，但是内容为空，这里肯定有问题。要解决这个问题，首先要了解 Apache 的启动机制及 httpd.pid 文件的作用。

httpd.pid 文件为文本文件，内容只有一行，记录了 httpd 进程的 pid。通过 cat 命令可以看到该文件的内容，通过这个 pid 文件可以防止进程启动多个副本。只有能获得 pid 文件的进程才能正常启动并把自身的 pid 写入该文件中。同一个程序的多余进程则自动退出。同时，httpd.pid 文件在 Apache 正常启动时创建，正常关闭时自动删除，Apache 在启动时会查找 httpd.pid 文件是否存在，如果文件不存在就创建此文件，将 Apache 启动的进程 ID 写入 httpd.pid 中，并提示启动成功。如果文件存在但内容为空，那么就会出现“(20014)Internal error”的错误了。

在这个案例中，httpd.pid 文件存在，但是内容为空，这个现象可能是磁盘空间耗尽导致的，也可能是系统突然断电导致的，总之导致这个问题的原因有很多种，这里并不打算深究，因为找到解决问题的办法是我们的目的。

3. 解决问题

解决这个问题有两个办法：一是直接删除 httpd.pid 这个空文件，二是将这个文件写入一个数字 ID 值，操作如下：

```

linux:~ # echo "28976">>/usr/local/apache2/logs/httpd.pid
linux:~ # more /usr/local/apache2/logs/httpd.pid
28976

```

接着，再次启动 Apache：

```
linux:~ # /usr/local/apache2/bin/apachectl start
```

这次 Apache 可以正常启动了，此时查看 httpd.pid 文件内容，信息如下：

```
linux:~ # more /usr/local/apache2/logs/httpd.pid
7789
```

可以看到，Apache 成功启动后，已经自动获得了一个新的 pid 值，进程间通信就以这个 pid 进行。Apache 在启动后，保持这个 pid 值不变，直到下次重新启动 Apache 才更新 httpd.pid 的值。

5.2.3 “could not bind to address 0.0.0.0:80” 错误与解决方法

1. 错误现象

客户的一台 Web 服务器是基于 Apache+JK+Tomcat 构建的一个电商平台，在机器更换硬件重新启动后，客户反映 Apache 无法启动，但是 Tomcat 可以启动。启动 Apache 的错误信息如图 5-23 所示。

```
[www@cloud1 ~]$ /usr/local/apache2/bin/apachectl start
(13)Permission denied: make_sock: could not bind to address [::]:80
(13)Permission denied: make_sock: could not bind to address 0.0.0.0:80
no listening sockets available, shutting down
Unable to open logs
```

图 5-23 Apache 无法启动错误现象

于是又检查了操作用户和 Apache 监听的端口，如图 5-24 所示。

```
[www@cloud1 apache2]$ whoami
www
[www@cloud1 apache2]$ cat /usr/local/apache2/conf/httpd.conf |grep Listen
# Listen: Allows you to bind Apache to specific IP addresses and/or
# Change this to Listen on specific IP addresses as shown below to
#Listen 12.34.56.78:80
Listen 80
```

图 5-24 查看 Apache 的管理用户和监听端口

从输出可知，Apache 的启动用户是 www，监听端口为 80，接着查看到 /usr/local/apache2 目录所有文件和目录的权限都是 www，看来不是读写权限的问题，于是继续排查，这里更换了 Apache 的监听端口，将其改为 8000，看能否成功启动，操作如图 5-25 所示。

```
[www@cloud1 apache2]$ sed -i 's/Listen 80/Listen 8000/' /usr/local/apache2/conf/httpd.conf
[www@cloud1 apache2]$ cat /usr/local/apache2/conf/httpd.conf |grep Listen
# Listen: Allows you to bind Apache to specific IP addresses and/or
# Change this to Listen on specific IP addresses as shown below to
#Listen 12.34.56.78:80
Listen 8000
[www@cloud1 apache2]$ /usr/local/apache2/bin/apachectl start
[www@cloud1 apache2]$ ps -ef|grep httpd
www      31756      1   0 18:39 ?        00:00:00 /usr/local/apache2/bin/httpd -k start
www      31758 30870   0 18:39 pts/4    00:00:00 grep httpd
```

图 5-25 修改 Apache 监听端口为 8000 后重启 Apache 服务

虽然这次启动没报错,但是根据 httpd 进程的状态来看,启动应该还有问题,于是查看 Apache 启动日志,如图 5-26 所示。

```
[Mon Sep 16 18:39:40 2013] [warn] pid file /usr/local/apache2/logs/httpd.pid overwritten -- Unclean shutdown of previous Apache run?
[Mon Sep 16 18:39:40 2013] [error] (13)Permission denied: could not create /usr/local/apache2/logs/httpd.pid
[Mon Sep 16 18:39:40 2013] [error] httpd: could not log pid to file /usr/local/apache2/logs/httpd.pid
```

图 5-26 Apache 启动错误信息

从日志输出看,果然存在问题,通过日志基本判断是 Apache 的 pid 文件无权限导致的。接着检查 httpd.pid 文件的权限,操作如下:

```
[root@cloud1 logs]# ll /usr/local/apache2/logs/httpd.pid
-rw-r--r-- 1 root www      6 Sep 16 17:33 /usr/local/apache2/logs/httpd.pid
```

从输出可知,httpd.pid 权限的属主为 root,将其修改为 www 用户,操作如下:

```
[root@cloud1 logs]# chown www /usr/local/apache2/logs/httpd.pid
```

然后,重启 Apache,操作如图 5-27 所示。

```
[www@cloud1 apache2]$ /usr/local/apache2/bin/apachectl restart
httpd not running, trying to start
[www@cloud1 apache2]$ ps -ef|grep httpd
www      31756      1   0 18:39 ?        00:00:00 /usr/local/apache2/bin/httpd -k start
www      32408      1   0 18:43 ?        00:00:00 /usr/local/apache2/bin/httpd -k restart
www      32409 32408   0 18:43 ?        00:00:00 /usr/local/apache2/bin/httpd -k restart
www      32410 32408   0 18:43 ?        00:00:00 /usr/local/apache2/bin/httpd -k restart
www      32411 32408   0 18:43 ?        00:00:00 /usr/local/apache2/bin/httpd -k restart
www      32414 32408   0 18:43 ?        00:00:00 /usr/local/apache2/bin/httpd -k restart
www      32573 30870   0 18:43 pts/4    00:00:00 grep httpd
```

图 5-27 重启 Apache 服务并查看启动进程

可以看到,这次 Apache 启动成功了,看来 Apache 配置并无问题,通过 8000 端口可以启动,而通过 80 端口则无法启动,这是什么问题呢?

2. 解决思路

既然这个案例是与端口相关的,那么就需要了解下 Linux 系统中的端口。在 Linux 系统

下可用的端口范围是 1~65535，端口可分为三类，分别是公认端口、注册端口和动态端口。

□ 公认端口的范围为 0~1023，属于系统预留端口，主要用于绑定一些服务，例如 80 端口绑定的是 HTTP 服务，21 端口绑定的是 FTP 服务，22 端口绑定的是 sshd 服务等。对于公认端口，Linux 系统做了一些安全限制，那就是普通用户无法绑定这类端口，只有 root 用户才能绑定和使用公认端口。

□ 注册端口的范围是 1024~49151，主要用于分配给用户进程或应用程序，这些进程主要是用户自定义安装的一些应用程序。

□ 动态端口的范围是 49152~65535，动态分配是指当一个系统进程或应用程序需要进行网络通信时，它就向主机申请一个端口，主机从可用的端口号中分配一个供它使用。当这个进程关闭时，同时也就释放了所占用的这个端口。

通过以上对端口的介绍，这个案例描述的问题也就明确了，在上面的错误现象中，普通用户 www 无法启动 Apache 的 80 端口，就是因为 80 端口属于公认端口，普通用户无权绑定，而 8000 端口属于注册端口，普通用户可以自由使用，这就是此案例要查找的原因。

3. 解决问题

如何使用 Apache 的 80 端口呢？这里提供两种方法，分别是：

- 1) 将 Apache 以 root 用户启动即可，这是最简单的方法。
- 2) 修改 Apache 目录下 httpd 文件的 SUID 属性。

第一种方法实现简单，但是有安全问题，如果黑客入侵了 80 端口，那么他也就拥有了 root 权限，因此，不推荐使用第一种方法，因为保证程序的安全才是根本。这里简单介绍一下第二种方法的实现过程。首先通过 root 用户进行如下授权，如图 5-28 所示。

```
[www@cloud1 apache2]$ chmod u+s /usr/local/apache2/bin/httpd
[www@cloud1 apache2]$ chown root /usr/local/apache2/bin/httpd
```

图 5-28 对 Apache 的 httpd 文件进行授权

然后在 www 用户下再次启动 Apache，可以看到这次启动正常了，如图 5-29 所示。

```
[www@cloud1 apache2]$ ps -ef|grep httpd
root    12892    1    0 19:46 ?        00:00:00 /usr/local/apache2/bin/httpd -k start
www     12893  12892    0 19:46 ?        00:00:00 /usr/local/apache2/bin/httpd -k start
www     12894  12892    0 19:46 ?        00:00:00 /usr/local/apache2/bin/httpd -k start
www     12895  12892    1 19:46 ?        00:00:00 /usr/local/apache2/bin/httpd -k start
www     12896  12892    0 19:46 ?        00:00:00 /usr/local/apache2/bin/httpd -k start
www     12983  30870    0 19:46 pts/4    00:00:00 grep httpd
```

图 5-29 通过 www 用户重启来启动 Apache 服务

从这个输出可以看出，其实 Apache 还是在 root 用户下启动的，上面的修改只不过是保

证普通用户可以正常启动 Apache 而已。另外,启动的 httpd 进程对应的用户除了 root,还有 www 用户,这其实是一种父进程和子进程的关系,父进程由 root 用户启动后,会派生出多个子进程,而这些子进程的启动用户是可定义的,可以在配置文件 httpd.conf 的如下选项中修改:

```
User www
Group www
```

至此,这个案例得到完整剖析!

5.3 因 NAS 存储故障引起的 Linux 系统恢复案例

5.3.1 故障现象描述

NAS 操作系统内核为 Linux,自带的存储有 16 块硬盘,总共分两组,每组都做了 RAID5, Linux 操作系统无法正常启动,在服务启动到 cups 那里就停止了,按 Ctrl+C 组合键强制断开也没有响应,硬盘状态都是正常的,没有报警或警告现象。

5.3.2 问题判断思路

通过上面这些现象,首先判断 NAS 硬件应该没问题, NAS 存储盘也应该正常,现在 Linux 无法启动,应该是 Linux 系统本身存在问题,因此,首先从 Linux 系统入手进行排查。

5.3.3 问题处理过程

1. 第一次处理过程

NAS 系统本身就是一个 Linux 内核装载了一个文件系统管理软件。管理软件可以对系统磁盘、系统服务、文件系统等进行管理 and 操作。在正常情况下,基于 Linux 内核的 NAS 系统应该启动到 init3 或 init5 模式下,由于 NAS 仅用了 Linux 一个内核模块和几个简单服务,因此判断 NAS 下的 Linux 系统肯定是启动到 init 3 模式下,那么既然现在无法启动到多用户字符界面下,何不让 Linux 直接进入单用户 (init 1) 模式下呢?因为单用户模式下仅仅启用系统所必需的几个服务,而 cups 服务是应用程序级别的,肯定不会在 init 1 模式下启动,这样就避开了 cups 无法启动的问题,所以,下面的工作就是要进入 Linux 的单用户模式下。

很多的 Linux 发行版本都可以在启动的引导界面通过相关的设置进入单用户模式下,

通过查看 NAS 的启动过程，基本判断这个 Linux 系统与 RHEL/CentOS 发行版极为类似，因此，通过 RHEL/CentOS 进入单用户模式下试一试。

RHEL/CentOS 进入单用户模式很简单，就是在系统启动到引导欢迎界面时按 e 键，然后编辑正确的内核引导选项，在最后面加上 “single” 选项，最后直接按 b 键即可进入单用户模式了。

接下来，重新启动 NAS，然后硬件自检，接着开始启动 Linux，一直在等待这个 NAS 的启动欢迎界面，但是欢迎界面一直没出现，就直接进入内核镜像开始加载内核阶段了。没有内核引导界面，如何进入单用户呢？经过简单思考，还是决定在硬件检测完毕后直接按 e 键，奇迹出现了，真的可以，NAS 进入了内核引导界面！通过简单观察，发现第二个正是要引导的内核选项，于是移动键盘中的上下键，选择这个内核，然后按 e 键，进入内核引导编辑界面，在这行的最后面输入 “single”，然后按回车键，返回上个界面，接着按 b 键开始进行单用户引导，经过 1 分钟的时间，系统如愿以偿地进入单用户模式下的 shell 命令行。

进入单用户模式后，能做的事情就很多了，首先要做的就是将 cups 服务在多用户模式下自启动关闭，执行命令如下：

```
chkconfig --level 35 cups off
```

执行成功后，重启系统进入多用户模式下，看看系统能否正常启动。

2. 第二次处理过程

将 cups 服务开机自启动关闭后，重启 NAS，发现问题依旧，NAS 还是在启动到 cups 服务那里停止了，难道上面的命令没有执行成功吗？明明已经禁止了 cups 服务启动了，怎么还是启动了呢？于是，继续重启 NAS，再次进入单用户模式下，看看问题究竟出在哪里了。

进入单用户后，再次执行 chkconfig 命令，依旧可以成功，难道是 cups 服务有问题，先看看配置文件，执行如下命令：

```
vi /etc/cups/cupsd.conf
```

在这里发现了一个问题，在通过 vi 命令打开 cupsd.conf 时，提示 “Write file in swap”，文件明明真实存在，怎么说在虚拟内存中呢？经过思考，只有一种可能，NAS 设备的 Linux 系统分区应该没有正确挂载，导致在进入单用户的时候，所有文件都存储在虚拟内存中，要验证这一点非常简单，执行 “df” 命令查看即可，如图 5-30 所示。

```
[root@NASserver ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/            11G   4.3G   5.8G  43% /
tmpfs            593M   4.0K   593M   1% /dev/shm
```

图 5-30 查看 NAS 挂载磁盘分区情况

从这里可以看出，Linux 的系统分区并未挂载，通过“fdisk -l”检查磁盘分区状态，输出如图 5-31 所示。

通过查看输出可知，NAS 的系统盘是 /dev/sda，仅划分了 /dev/sda1 和 /dev/sda2 两个系统分区，而数据磁盘是经过做 RAID5 完成的，在系统上的设备标识分别是 /dev/sdb1 和 /dev/sdc1。由于单用户模式下默认没有挂载任何 NAS 磁盘，这里尝试手动挂载 NAS 的系统盘，执行如下命令：

```
[root@NASserver ~]# mount /dev/sda2 /mnt
[root@NASserver ~]# mount /dev/sda1 /opt
```

```
[root@NASserver ~]# fdisk -l

Disk /dev/sda: 85.8 GB, 85899345920 bytes
255 heads, 63 sectors/track, 10443 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks    Id System
/dev/sda1  *           1           13       104391    83  Linux
/dev/sda2             14        10443     83778975    8e  Linux

Disk /dev/sdb: 1999.8 GB, 1999844147200 bytes
255 heads, 63 sectors/track, 243133 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks    Id System
/dev/sdb1  *           1        243133    1952965791    83  Linux

Disk /dev/sdc: 1999.8 GB, 1999844147200 bytes
255 heads, 63 sectors/track, 243133 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks    Id System
/dev/sdc1  *           1        243133    1952965791    83  Linux
```

图 5-31 查看 NAS 磁盘分区情况

这里的 /mnt、/opt 是随意挂载的目录，也可以挂载到其他空目录下，完成挂载后分别进入这两个目录看看其中是否有内容，如图 5-32 和图 5-33 所示。

```
[root@NASserver opt]# cd /mnt
[root@NASserver mnt]# ls -al
drwxr-xr-x  2 root  root   4096 Apr 12 14:23 bin
drwxr-xr-x 11 root  root   4420 Jul  6 14:37 dev
drwxr-xr-x  5 root  root   4096 Jul 18 15:11 home
drwxr-xr-x 97 root  root  12288 Sep 18 04:26 etc
drwxrwxrwt  8 root  root   4096 Sep 21 04:02 tmp
drwxr-xr-x 18 root  root   4096 Jul 18 21:27 usr
drwxr-xr-x 22 root  root   4096 Apr 12 13:09 var
```

图 5-32 查看 /dev/sda2 磁盘标识下的内容


```

[root@NASserver ~]# cd /opt
[root@NASserver opt]# ls -al
total 6226
drwxr-xr-x  4 root root    1024 Apr 12 13:10 .
drwxr-xr-x 36 root root    4096 Jul  6 14:36 ..
-rw-r--r--  1 root root   67541 Feb 22  2012 config-2.6.18-308.el5
drwxr-xr-x  2 root root    1024 Apr 12 13:11 grub
-rw-----  1 root root 2651208 Apr 12 13:10 initrd-2.6.18-308.el5.img
drwx-----  2 root root    12288 Apr 12 13:06 lost+found
-rw-r--r--  1 root root    80032 Mar 13  2009 message
-rw-r--r--  1 root root   116635 Feb 22  2012 symvers-2.6.18-308.el5.gz
-rw-r--r--  1 root root 1275921 Feb 22  2012 System.map-2.6.18-308.el5
-rw-r--r--  1 root root 2115772 Feb 22  2012 vmlinuz-2.6.18-308.el5

```

图 5-33 查看 /dev/sda1 磁盘标识下的内容

通过查看这两个内容,初步判断, /dev/sda2 分区应该是 Linux 的根分区,而 /dev/sda1 应该是 /boot 分区。现在分区已经挂载上去了,再次执行 df 命令查看挂载情况,如图 5-34 所示。

```

[root@NASserver ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/           11G  4.3G  5.8G  43% /
tmpfs           8G    0    8G   0% /dev/shm
/dev/sda2       72G  72G   0 100% /mnt
/dev/sda1       99M  20M  75M  21% /opt

```

图 5-34 查看 NAS 系统盘的磁盘空间

到这里为止,发现问题了。/dev/sda2 磁盘分区已经没有可用的磁盘空间了,而这个分区刚好是 NAS 系统的根分区,根分区没有空间了,那么系统启动肯定就出问题了。

下面再把思路转到前面介绍的案例中,由于系统 cups 服务在启动的时候会写启动日志到根分区,而根分区没有空间了,因此也就无法写日志了,由此导致的结果就是 cups 服务无法启动,这就解释了此案例中 NAS 系统每次启动到 cups 服务就停止的原因。

5.3.4 解决问题

由于 NAS 系统只有根分区和 /boot 分区,因此系统产生的相关日志都会存储在根分区中,现在根分区满了,首先可以清理的就是 /var 目录下的系统相关日志文件,通常可以清理的目录有 /var/log。执行如下命令查看 /var/log 日志目录占据磁盘空间大小:

```

[root@NASserver ~]# du -sh /var/log
50.1G    /var/log

```

通过命令输出发现 /var/log 目录占据了根分区仅 70% 的空间,清理这个目录下的日志文件即可释放大部分根分区空间,清理完毕后重启 NAS 系统,发现系统 cups 服务能正常启动了, NAS 服务也启动正常了。

第三部分 *Part 3*

自动化运维篇

6.1 并行 SSH 运维工具 pssh

随着企业 IT 基础设施规模不断扩张，业务迅速扩大，企业管理人员要管理的服务器和业务系统也迅速增加，从之前的几十台增加到上百台甚至几千台。面对这么多的服务器，要执行相同或类似配置操作，一台一台地部署显然是不可取的，而通过 shell 脚本来完成，效率又十分低下。此时就需要一些自动化的运维工具来接管这些工作。本章重点介绍三个轻量级运维利器，分别是 pssh、pdsh 和 musssh。这些工具小巧易用，功能强大，可以为自动化运维提供强有力的支持。

6.1.1 pssh 应用场景

- 第6章 轻量级运维利器 pssh、pdsh 和 musssh
- 第7章 分布式监控系统 Ganglia
- 第8章 基于 nagios 的分布式监控报警平台 Centreon
- 第9章 通过 Ganglia 与 Centreon 构建智能化监控报警平台

要使用 pssh 工具，首先要建立本地主机和远程所有服务器上的普通用户或 root 用户之间的单向信任，也就是要在本地主机和远程所有服务器上部署公钥认证访问，创建本地主机和远程主机之间的单向信任非常简单，下面以创建本地主机与 opnsbl 和所有远端主机上 opnsbl 用户之间的信任为例介绍配置的基本过程。

轻量级运维利器 pssh、pdsh 和 mussh

6.1 并行 SSH 运维工具 pssh

随着企业 IT 基础架构规模不断扩张，业务也迅速扩大，企业运维人员要管理的服务器和业务系统也迅速增加，从之前的几十台增加到上百台甚至几千台。面对这么多的服务器，要执行相同的系统配置操作，一台一台地部署显然是不现实的，而通过写 shell 脚本来完成，效率又十分低下，此时就需要一些自动化运维工具来批量完成。本章重点介绍三个轻量级运维利器，分别是 pssh、pdsh 和 mussh，这三个工具小巧、使用简单，但功能强大，可以为自动化运维提供强有力的支持。

6.1.1 pssh 应用场景

pssh 的全称是 parallel-ssh，是一个用 Python 编写的可以并发在多台服务器上批量执行命令的工具，它支持文件并行复制、远程并行执行命令、杀掉远程主机上的进程等。其中，文件并行复制是 pssh 核心功能，也是同类工具中最大的亮点，因此，要批量在远程主机上传、下载文件时，最好选用 pssh 这个服务器批量管理工具。

要使用 pssh 工具包，必须保证本地主机和要管理的远程主机之间的单向信任，也就是要在本地主机和远程所有服务器上配置密钥认证访问。创建本地主机和远程主机之间的单向信任非常简单，下面以创建本地用户 opsuser 和所有远程主机上 opsuser 用户之间的信任为例介绍配置的基本过程。

1. 在本地主机上创建 RSA 密钥和公钥

- 1) 在本地主机上以 opsuser 用户登录。
- 2) 在 opsuser 用户的根目录内创建 .ssh 目录并设置读取权限。

```
[opsuser@server ~]$ mkdir ~/.ssh
[opsuser@server ~]$ chmod 700 ~/.ssh
```

- 3) 使用 ssh-keygen 命令生成基于 SSH 协议的 RSA 密钥。

```
[opsuser@server ~]$ cd ~/.ssh
[opsuser@server ~]$ ssh-keygen -t rsa
```

在提示保存私钥 (key) 和公钥 (public key) 的位置时, 选择默认值, 然后依次按回车键即可。

2. 整合公钥文件

在本地主机上以 opsuser 用户登录, 执行如下操作:

```
[opsuser@server ~]$ cd ~/.ssh
[opsuser@server ssh]$ cat /home/opsuser/.ssh/id_rsa.pub > authorized_keys
[opsuser@server ssh]$ chmod 600 ~/.ssh/authorized_keys
[opsuser@server ssh]$ scp authorized_keys user001:/home/opsuser/.ssh/
```

这个操作过程是将本地节点生成的公钥文件整合为一个 authorized_keys 文件, 然后进行授权, 并将 authorized_keys 复制到所有远程主机上。这里以远程主机 user001 为例。

3. 测试 SSH 密钥认证

在本地主机上执行如下命令:

```
[opsuser@server ~]$ ssh user001 date
```

这里以 user001 主机为例, 其他主机依此类推, 如果不需要输入密码就出现系统当前日期, 说明 SSH 单向信任已经配置成功了。

6.1.2 pssh 的安装与用法

pssh 要求 Python 版本大于 2.4 即可。大家可以到 <http://code.google.com/p/parallel-ssh/> 下载 pssh 的各种版本, 目前最新版本是 pssh-2.3.1。这里下载的是 pssh-2.3.1.tar.gz, 安装过程如下:

```
[root@server ~]# tar zxvf pssh-2.3.1.tar.gz
[root@server ~]# cd pssh-2.3.1
```



```
[root@server pssh-2.3.1]# python setup.py install
```

安装完成的 pssh 工具包附带了 5 个主程序，分别是：

- ❑ parallel ssh (pssh)，在远程多台主机上并行运行命令。
- ❑ parallel scp (pscp)，把文件并行复制到远程多台主机上，类似于 scp 命令。
- ❑ parallel rsync (prsync)，使用 rsync 协议将文件从本地主机同步到远程多台主机上。
- ❑ parallel nuke (pnuke)，在远程多台主机上并行 killall 某一进程。
- ❑ parallel slurp (pslurp)，把文件从远程多台主机复制到本地主机，与 pscp 命令相反。

完成 pssh 安装后，执行“pssh --help”命令即可显示 pssh 命令的详细用法，表 6-1 是 pssh 常用的参数以及语法介绍。

表 6-1 pssh 命令的常用参数以及含义

参数	含义
-h HOST_FILE	此参数后面跟一个远程主机列表文件，文件内容格式为 [user@]host[:port]，每行一个，也可省略用户名和端口号，省略时默认用户为执行 pssh 命令所在用户，默认端口为 22。例如： test@192.168.12.189:9527 192.168.16.223
-H HOST_STRING	此参数后面跟一个远程主机名或 IP 地址，格式为：[user@]host[:port]，例如： pssh -H 192.168.16.166 date
-l USER	此参数指定远程主机上的用户名
-p PAR	指定 pssh 最大并行线程数，例如： pssh -p 10
-o OUTDIR	将输出的内容重定向到一个指定的文件中
-O OPTION	指定 SSH 参数的具体配置，可以参照 ssh_config 文件的配置参数，例如： pssh -O StrictHostKeyChecking=no
-e ERRDIR	将执行错误重定向到一个指定的文件中
-t TIMEOUT	设置命令执行的超时时间
-A	提示输入密码，并把密码传递给 SSH 服务
-x ARGS	用于传递 ssh 命令的一些参数，每个参数用引号括起来，当需要传递多个 ssh 命令参数时，不同参数之间用空格分开 例如： pssh -x "-l user001" "-p 22" 其中“-l”和“-p”都是 ssh 命令的参数
-X ARG	与“-x”用法一样，不过“-X”只能指定一个 ssh 命令参数
-i	在远程主机上执行命令完成后显示标准输出和标准错误
-P	在执行远程命令时，输出执行结果

这里重点介绍了 pssh 命令的一些常用参数，而其他四个命令 pscp、prsync、pnuke 和 pslurp 的参数含义与用法与 pssh 命令基本相同，因此这里不再过多介绍。

6.1.3 pssh 应用实例

pssh 命令的使用非常灵活，可以根据实际的应用需要选择对应的命令工具，例如，要在远程主机上批量执行一些信息统计命令，就可以通过 pssh 命令来完成。而要将一些文件传输到远程的多台主机上，或者将远程主机的某些文件复制到本地主机时，就需要使用 pscp 命令和 pslurp 命令了，如果要传输某些文件夹到远程多台主机上时，就必须选择 prsync 命令了。

1. pssh 批量查看远程主机信息

通过 pssh 命令查看远程主机 user002 和 user003 上面的时间信息，实现过程如图 6-1 所示。

```
[opsuser@server ~]$ pssh -H opsuser@user002.bestbook.com:9529 -P date
user002.bestbook.com: Sat Jan  4 21:49:06 CST 2014
[1] 21:49:06 [SUCCESS] opsuser@user002.bestbook.com:9529

[opsuser@server ~]$ pssh -H user003.bestbook.com -i date
[1] 21:49:10 [SUCCESS] user003.bestbook.com
Sat Jan  4 21:49:10 CST 2014
```

图 6-1 通过 pssh 命令查看 user002 和 user003 服务器上的时间信息

在这个操作中，通过“-H”参数指定远程的单台主机，另外，通过这个操作可以很清楚地看到“-P”参数与“-i”参数的差别。

pssh 命令更多的用途是批量管理服务器，下面的例子演示的就是批量查看远程多台服务器的负载状况，如图 6-2 和图 6-3 所示。

```
[opsuser@server ~]$ more /etc/pssh/hosts
user001.bestbook.com
user002.bestbook.com
user003.bestbook.com
.....
```

图 6-2 远程服务器列表

```
[opsuser@server ~]$ pssh -i -O "StrictHostKeyChecking=no" -h /etc/pssh/hosts "uptime"
[1] 22:24:32 [SUCCESS] user004.bestbook.com
22:24:32 up 204 days, 5:22, 1 user, load average: 2.23, 3.58, 2.08
[2] 22:24:32 [SUCCESS] user002.bestbook.com
22:24:32 up 210 days, 19:54, 0 users, load average: 3.35, 5.04, 3.93
[3] 22:24:32 [SUCCESS] user008.bestbook.com
22:24:32 up 210 days, 19:53, 0 users, load average: 5.90, 6.50, 4.52
[4] 22:24:32 [SUCCESS] user001.bestbook.com
22:24:32 up 210 days, 19:54, 4 users, load average: 2.35, 3.45, 1.99
.....
```

图 6-3 通过 pssh 获取的远程服务器负载信息

这个操作中用到了“-O”参数，后面跟的“StrictHostKeyChecking=no”是 sshd 服务的配置文件 ssh_config 中的一个选项，通过设置这个参数，可以让远程主机自动接受本地主机的 hostkey，而不用每次都要手动输入 yes。后面还用到了“-h”参数，通过这个参数

可以指定一个需要连接的服务器列表，文件内容如图 6-2 所示。在这个文件中省略了用户名和连接端口，那么 pssh 将自动使用当前用户 opsuser 和默认端口 22。

从图 6-3 的输出结果看，获取的远程主机负载信息不是按照服务器列表的顺序显示的，这刚好说明 pssh 的并发执行特性。

pssh 可调用 tar 命令解压远程主机上的文件，例如：

```
[opsuser@server ~]$ pssh -i -h /etc/pssh/hosts "tar -zxvf hadoop-2.0.0-cdh4.5.0.tar.gz"
```

在这个例子中，pssh 默认解压的是远程主机上 /home/opsuser/hadoop-2.0.0-cdh4.5.0.tar.gz 文件，而要解压 root 用户下的文件，可以执行如下操作：

```
[opsuser@server ~]$ pssh -i -h /etc/pssh/hosts \  
> "sudo "tar -zxvf /mnt/hadoop-2.0.0-cdh4.5.0.tar.gz -C /mnt" "
```

在这个例子中，pssh 调用了“sudo”命令，因此，要操作具有 root 权限的文件时，opsuser 用户在远程主机上必须开通可通过 sudo 命令切换到 root 用户的权限。

pssh 也可批量删除远程主机上指定的文件或目录，例如：

```
[opsuser@server ~]$ pssh -i -h /etc/pssh/hosts "sudo "rm -rf /mnt/hadoop-2.0.0-cdh4.5.0" "
```

类似的例子有很多，比如通过 pssh 还可以在远程主机上安装软件、启动系统服务等，看下面两个例子：

```
[opsuser@server ~]$ pssh -i -h /etc/pssh/hosts "sudo "yum -y install pssh" "  
[opsuser@server ~]$ pssh -i -h /etc/pssh/hosts "sudo "/etc/init.d/gmond start" "
```

下面这个操作是 pssh 的综合应用实例：

```
[opsuser@server ~]$ pssh -i -x "-l opsuser" "-p 9529" -h /etc/pssh/hosts \  
> -o /etc/pssh/info "uptime;uname -a"
```

这个实例用到了“-x”参数，分别调用了 ssh 命令的两个参数“-l”和“-p”，用于指定在远程主机上登录的用户名和 SSH 端口，而“-o”参数指定将输出结果存放到 /etc/pssh/info 目录中。在此实例的最后指定了要在远程主机上执行的命令，可以指定多个命令，每个命令之间用分号隔开。

2. pscp 与 pslurp 应用实例

pscp 命令的主要作用是将本地文件并行地复制到远程多台主机上，而 pslurp 是把文件从远程多台主机复制到本地主机，这两个命令的功能刚好相反。在运维工作中，需要进行文件批量传送时，这两个命令非常有用。

首先看 pscp 的两个应用实例，如图 6-4 所示。

```
[opsuser@server ~]$ pscp -h /etc/pssh/hosts /etc/ssh/ssh_config /tmp
[opsuser@server ~]$ pscp -h /etc/pssh/hosts -r /etc/httpd/conf /tmp
```

图 6-4 pscp 应用实例

图 6-4 中第一个例子是将本地主机上的 `/etc/ssh/ssh_config` 文件复制到远程主机的 `/tmp` 目录下，这是 `pscp` 最简单的用法，如果要复制的文件比较多，一个一个执行就非常麻烦，此时可以通过复制目录的方式实现。第二个例子就是将本地的 `/etc/httpd/conf` 目录复制到远程主机的 `/tmp` 目录下，在进行目录复制时，用到了“`-r`”参数，表示递归地复制指定目录下的所有文件。这里需要注意权限问题，指定远程主机上的目标路径一定是当前用户可读写，否则会发生错误。

完成复制后，在远程主机上查看复制过来的目录，结构如下：

```
[opsuser@user001 ~]$ ls -ld /tmp/conf
drwxr-xr-x 2 opsuser opsuser 4096 Jan  5 16:06 /tmp/conf
```

接下来再看 `pslurp` 的两个应用实例，如图 6-5 所示。

```
[opsuser@server ~]$ pslurp -h /etc/pssh/hosts -L /home/opsuser/test \
> /home/opsuser/gmond.conf gmond1.conf
[opsuser@server ~]$ pslurp -h /etc/pssh/hosts -r -L /home/opsuser/test \
> /home/opsuser/gmond gmond1
```

图 6-5 pslurp 应用实例

图 6-5 中第一个例子是将所有远程主机上的 `/home/opsuser/gmond.conf` 文件复制到本地主机的 `/home/opsuser/test` 目录下，并将文件改名为 `gmond1.conf`，这里面用到了“`-L`”参数，用来指定本地主机路径，这个路径是个目录，用于存储从远程主机传输过来的文件。第二个例子是将所有远程主机上的 `/home/opsuser/gmond` 目录复制到本地主机的 `/home/opsuser/test` 目录下，并将复制过来的目录改名为 `gmond1`，这里面也用到了“`-r`”和“`-L`”参数，需要注意这两个参数的顺序不能颠倒。

完成数据复制后，在本地主机的 `/home/opsuser/test` 目录下存放着所有以远程主机命名的目录，这里以远程 `user001` 主机为例，查看复制过来的文件，结果如图 6-6 所示。

```
[opsuser@server test]$ pwd
/home/opsuser/test
[opsuser@server test]$ tree user001.bestbook.com
user001.bestbook.com
|-- gmond1
|   |-- gmond.conf
|   |-- gmond1.conf
1 directory, 2 files
```

图 6-6 pslurp 复制远程文件结构图

3. prsync 与 pnuke 应用实例

prsync 的主要作用是通过 rsync 协议将文件或目录从本地主机同步到远程多个主机上, 先看 prsync 的两个应用实例, 如图 6-7 所示。

```
[opsuser@server ~]$ prsync -h /etc/pssh/hosts -l opsuser -a -r /etc/httpd /tmp
[opsuser@server ~]$ ls -ld /tmp/httpd
drwxr-xr-x 4 opsuser opsuser 4096 Mar 12 2012 /tmp/httpd
[opsuser@server ~]$ prsync -h /etc/pssh/hosts -l opsuser -az -r /var/log/httpd /tmp
```

图 6-7 prsync 应用实例

图 6-7 的第一个实例中使用了“-l”、“-a”和“-r”参数, 其中“-l”用于指定远程主机上的用户, “-r”用于递归复制指定目录下的所有文件, 这两个参数前面已经介绍过, 而“-a”参数可以维持文件的属性值不变, 例如文件的创建时间、修改时间、读写权限等, 这个参数在做目录复制时非常有用, 建议使用。

第二个实例中又引入了“-z”参数, 这是一个压缩传输参数, 在低带宽环境下, 或者在对网络带宽有要求、传输文件压缩率比较大时使用。而在带宽充足或传输的文件比较大时, 不推荐使用。

pnuke 的主要作用是在远程多主机上并行杀掉某一进程, 相当于 killall 命令。它的用法非常简单, 最简单的用法如下:

```
[opsuser@server ~]$ pnuke -h /etc/pssh/hosts httpd
```

这个命令将在远程所有主机上并行关闭 httpd 服务, pnuke 的作用类似于 killall, 而后跟的 httpd 是服务名, 其实只要通过 killall 命令能关闭的服务, 都可以通过 pnuke 来批量完成。

6.2 并行分布式运维工具 pdsh

pdsh 的全称是 parallel distributed shell, 与 pssh 类似, pdsh 可并行执行对远程目标主机的操作, 在有批量执行命令或分发任务的运维需求时, 使用这个命令可达到事半功倍的效果。同时, pdsh 还支持交互模式, 当要执行的命令不确定时, 可直接进入 pdsh 命令行, 非常方便。

6.2.1 pdsh 应用场景

pdsh 的应用场景与 pssh 基本相同, 都用于大批量服务器的配置、部署、文件复制等运维操作。在使用 pdsh 时, 仍需要配置本地主机和远程主机间的单向 SSH 信任。另外, pdsh

还附带了 `pdcp` 命令，此命令可以将本地文件批量复制到远程的多台主机上，这在大规模的文件分发环境下是非常有用的。

`pdsh` 可以通过多种方式在远程主机上运行命令，默认是 `rsh` 方式，另外也支持 `ssh`、`mrsh`、`qsh`、`mqsh`、`krb4`、`xcpu` 等多种 `rcmd` 模块，这可以在运行命令时通过参数指定。

6.2.2 `pdsh` 的安装与语法

1. `pdsh` 的安装过程

`pdsh` 的安装非常简单，有 `RPM` 包和源码包两种安装方式，读者可根据自己的喜好选择适合自己的安装方式。可以在 <http://code.google.com/p/pdsh/> 下载最新的源码包进行编译安装，目前最新版本为 `pdsh-2.29`，这里下载的源码包为 `pdsh-2.29.tar.bz2`。编译安装过程如下：

```
[root@server ~]# tar jxvf pdsh-2.29.tar.bz2
[root@server ~]# cd pdsh-2.29
[root@server pdsh-2.29]# ./configure --with-ssh --with-rsh --with-mrsh --with-mqshell \
> --with-qshell --with-dshgroups --with-machines=/etc/pdsh/machines
[root@server pdsh-2.29]# make
[root@server pdsh-2.29]# make install
```

在执行 `configure` 阶段，“`--with-ssh`”参数表示启用 `ssh` 模块，其他参数都有类似的含义，而“`--with-dshgroups`”表示启用主机组支持，启用此参数后，就可以将一组主机列表写入一个文件并放到 `~/dsh/group` 或 `/etc/dsh/group` 目录下，然后通过 `pdsh` 的“`-g`”参数进行调用。最后的参数“`--with-machines`”是“`--with-dshgroups`”参数的扩展，通过将所有要管理的主机列表都写入指定的 `/etc/pdsh/machines` 文件中，接着通过 `pdsh` 的“`-a`”参数调用，最终完成所有主机的便捷管理。

完成安装后，可以通过“`pdsh -V`”命令查看 `pdsh` 的版本号以及可使用的模块信息，操作如下：

```
[opsuser@server ~]$ pdsh -V
pdsh-2.29
rcmd modules: ssh,rsh,exec (default: rsh)
misc modules: machines,dshgroup
```

此外，也可以通过“`pdsh -L`”命令来显示当前所有加载的模块信息。在 `pdsh` 安装完成后，还有一个可用的工具 `pdcp`，后面将详细讲述 `pdsh` 和 `pdcp` 两个命令的用法。

2. `pdsh` 的语法介绍

安装 `pdsh` 完成后，通过执行“`pdsh -h`”和“`pdcp -h`”命令即可得到两个命令的完整

用法，由于两个命令的参数大同小异，因此这里以 pdsh 命令为主介绍一些常用的参数及含义。表 6-2 是 pdsh 常用的参数及其含义。

表 6-2 pdsh 常用的参数及含义

参数	含义
-w host,host	<p>指定远程主机，可以指定多台，每台主机用逗号隔开，host 可以是主机名也可以是 IP 地址。此参数非常灵活，常用的形式有：</p> <pre>pdsh -w ssh:user001,ssh:user002,ssh:user003 "date"</pre> <p>此命令用来查看 user001、user002、user003 主机上的时间，其中 ssh 表示在远程主机上执行命令的形式，默认是 rsh</p> <pre>pdsh -w ssh:user00[1-10] "date"</pre> <p>此命令用于在 user001 到 user0010 上执行 date 命令</p> <pre>pdsh -w ssh:user0[10-31],/1\$/ "uptime"</pre> <p>此命令在选择远程主机时使用了正则表达式，表示在 user010 到 user031 中选择以 1 结尾的主机名，即在 user011、user021、user031 上执行 uptime 命令</p>
-R	<p>指定使用 rcmd 的模块名，默认是 rsh。如果要选择 ssh，可以通过如下方式指定：</p> <pre>pdsh -R ssh -w user00[1-10] "date"</pre>
-l	<p>指定在远程主机上使用的用户名称。例如：</p> <pre>pdsh -R ssh -l opsuser -w user00[1-9] "date"</pre>
-x	<p>此参数用来排除某些或某台主机，例如：</p> <pre>pdsh -R ssh -l opsuser -w user00[1-9] -x user005,user007 "date"</pre>
-t	<p>指定连接远程主机的超时时间，以 s 为单位，默认是 10s，可以通过此参数修改默认值，例如：</p> <pre>pdsh -R ssh -w slave000[1-9] -t 15 "date"</pre>
-u	设置远程命令执行的超时时间，以 s 为单位，以 SSH 方式连接时，默认时间为无限
-f	设置同时连接到远程主机的数量
-N	此参数用来关闭远程主机所返回结果中的主机名显示
-a	通过此参数可以指定所有的远程主机，设置此参数后，pdsh 默认会查看 /etc/machines 文件中的主机列表，要改变此路径，在编译 pdsh 时通过 “--with-machines” 参数指定即可
-g	<p>此参数用来指定一组远程主机，在编译 pdsh 时可以通过 “--with-dshgroups” 参数来激活此选项，默认可以将一组主机列表写入一个文件中并放到本地主机的 ~/.dsh/group 或 /etc/dsh/group 目录下，这样就可以通过 “-g” 参数调用了。例如：</p> <pre>pdsh -R ssh -g userhosts "date"</pre> <p>其中 “userhosts” 是一个主机列表文件，可以将此文件放在 ~/.dsh/group 或 /etc/dsh/group 目录下</p>
-X	<p>此参数用来排除指定组内的所有主机，经常与 “-a” 参数一起使用。例如：</p> <pre>pdsh -R ssh -a -X userhosts "date"</pre>
-q	此参数可以列出 pdsh 执行时的一些配置信息
-V	此参数可以查看软件的版本信息以及可用的模块信息

6.2.3 pdsh 应用实例

在对远程主机的批量管理操作上，pdsh 功能非常强大，它比 pssh 应用更加灵活，原

则上只要设置了本地主机与远程主机的信任,就可以通过 `pdsh` 工具在远程主机上做任何事情。但是,这并不是说 `pdsh` 安全性低,相反,`pdsh` 支持多种访问远程主机的方式,在对安全性要求较高的环境中,可以选择 `rsh` 来访问远程主机,不过本章的内容都是以 `ssh` 的访问方式来介绍的。

1. `pdsh` 批量统计主机信息

先列举一个 `pdsh` 命令应用的最简单例子。例如,要统计远程主机 192.168.16.1、192.168.16.2、192.168.16.3 的主机名信息,可以执行如下命令,如图 6-8 所示。

```
[opsuser@server ~]$ pdsh -w ssh:192.168.16.[1-3] "uname -n"
192.168.16.2: user002.bestbook.com
192.168.16.3: user003.bestbook.com
192.168.16.1: user001.bestbook.com
```

图 6-8 使用 `pdsh` 命令统计远程主机的主机名信息

在这个实例中,使用了 `pdsh` 命令的“-w”参数,然后指定和远程主机通信的方式为 `ssh`,这里重点关注远程批量主机的写法。

如果要在指定的批量主机中排除某台或某几台主机不做统计,可以使用 `pdsh` 的“-x”参数,如图 6-9 所示。

```
[opsuser@server ~]$ pdsh -w ssh:user00[1-5] -x user003 "uptime"
user001: 21:42:37 up 212 days, 19:12, 4 users, load average: 4.81, 4.87, 5.16
user004: 21:42:37 up 206 days, 4:40, 1 user, load average: 5.30, 5.67, 6.36
user005: 21:42:37 up 212 days, 19:12, 0 users, load average: 5.57, 5.42, 5.80
user002: 21:42:37 up 212 days, 19:12, 0 users, load average: 5.46, 5.56, 5.74
```

图 6-9 `pdsh` 命令在排除远程主机时的用法

这个例子是统计每台远程主机的负载状态,从输出结果可以看出,远程主机 `user003` 已经被排除了。

`pdsh` 命令在远程主机过滤方面功能非常强大,它可以支持正则匹配,图 6-10 展示的是匹配主机名以 1 结尾的主机。

```
[opsuser@server ~]$ pdsh -w ssh:user00[10-31],1$/ "uptime"
user0021: 21:41:42 up 119 days, 4:02, 0 users, load average: 5.17, 5.36, 5.54
user0011: 21:41:42 up 212 days, 19:07, 0 users, load average: 5.64, 5.43, 6.03
user0031: 21:41:42 up 212 days, 19:05, 1 user, load average: 7.06, 6.08, 5.77
```

图 6-10 在 `pdsh` 命令中通过正则匹配远程主机

从输出来看,从 `user0010` 到 `user0031` 的主机中,匹配到的主机有 `user0021`、`user0011` 和 `user0031`,刚好这三台主机的主机名都是以 1 结尾。这里重点关注正则匹配的写法。

有时候需要批量管理所有的远程主机，而远程主机的主机名又不是规范的，此时可以将所有主机名写到一个文件中，然后通过 pdsh 命令的“-a”参数调用即可，非常方便，如图 6-11 所示。

```
[opsuser@server ~]$ more /etc/pdsh/machines
user001
user002
user0010
user0011
[opsuser@server ~]$ pdsh -R ssh -a uptime
user001: 00:38:31 up 212 days, 22:08, 5 users, load average: 9.12, 8.82, 8.87
user002: 00:38:31 up 212 days, 22:08, 0 users, load average: 7.76, 8.23, 8.46
user0010: 00:38:31 up 212 days, 22:04, 0 users, load average: 8.54, 9.12, 9.40
user0011: 00:38:31 up 212 days, 22:04, 0 users, load average: 8.98, 9.25, 9.17
```

图 6-11 在 pdsh 命令中通过“-a”参数匹配所有主机

在这个例子中，将所有远程主机的主机名都写到本地主机的 /etc/pdsh/machines 文件中，这个路径是在编译 pdsh 时通过“--with-machines”参数指定的。然后只需要指定“-a”参数，pdsh 命令就会自动去指定的文件中读写主机名信息。

在大规模运维环境中，服务器往往会根据业务系统的不同分为多个组，例如 Web 服务器组、数据库服务器组等。要批量管理这些服务器，也需要对这些服务器进行分组，可以将 Web 服务器组的所有主机名信息写入一个文件中，将数据库服务器组的所有主机名信息写入另一个文件中，然后通过 pdsh 命令的“-g”参数进行调用即可，如图 6-12 所示。

```
[opsuser@server ~]$ more /etc/dsh/group/userhosts
user001
user002
[opsuser@server ~]$ pdsh -R ssh -g userhosts uptime
user001: 00:36:44 up 212 days, 22:06, 5 users, load average: 9.42, 8.86, 8.90
user002: 00:36:44 up 212 days, 22:06, 0 users, load average: 8.25, 8.42, 8.54
```

图 6-12 在 pdsh 命令中通过“-g”参数对远程主机进行分组管理

要实现远程主机的分组管理，需要激活 pdsh 的 dshgroup 模块，也就是在编译 pdsh 的时候指定“--with-dshgroups”参数即可。激活这个功能后，就可以将不同用途的服务器进行分组，将主机名列表文件放到 ~/.dsh/group 或 /etc/dsh/group 目录下。在这个例子中，“-g”参数后面的 userhosts 就是分组后的主机名列表文件。

接着看一下 pdsh 命令参数中“-X”的用法，如图 6-13 所示。

```
[opsuser@server ~]$ pdsh -R ssh -a -X userhosts uptime
user0011: 00:39:25 up 212 days, 22:05, 0 users, load average: 8.91, 9.18, 9.40
user0010: 00:39:25 up 212 days, 22:05, 0 users, load average: 10.29, 9.55, 9.27
```

图 6-13 在 pdsh 命令中通过“-X”参数排除指定组内的所有主机

这个例子结合了上面例子的环境，使用了“-a”和“-X”两个参数，从输出结果来看，

证实了“-X”参数的功能，即排除指定组内的所有主机。

最后，再来测试一下 pdsh 如何在远程主机上执行命令，如图 6-14 所示。

```
[opsuser@server ~]$ pdsh -R ssh -g userhosts "rm -rf /home/opsuser/mysql"
[opsuser@server ~]$ pdsh -R ssh -g userhosts "sudo mkdir /mnt/test"
[opsuser@server ~]$ pdsh -R ssh -g userhosts "sudo /etc/init.d/gmond start"
slave0002: Starting GANGLIA gmond: [ OK ]
slave0001: Starting GANGLIA gmond: [ OK ]
```

图 6-14 pdsh 在远程主机上执行命令的方式

pdsh 命令在远程主机上执行命令的方式与 pssh 基本相同，对于有权限操作的文件，可以直接指定命令进行删除或创建，而对于没有权限操作的文件或服务，可以通过在命令前添加 sudo 命令的方式来提升操作权限。当然，这需要当前操作的用户具有通过 sudo 命令切换到 root 用户的权限。

pdsh 命令还支持交互模式，当需要执行的命令不确定时，可以通过交互模式进行，如图 6-15 所示。

```
[opsuser@server ~]$ pdsh -R ssh -w user001
pdsh>mkdir mysql
pdsh>cp /etc/my.conf .
pdsh> echo "user001.bestbook.com" >hosts
pdsh> cat hosts
user001: user001.bestbook.com
pdsh> ls -l
user001: total 12
user001: -rw-r--r-- 1 opsuser opsuser 21 Jan 8 22:49 hosts
user001: -rw-r--r-- 1 opsuser opsuser 1006 Jan 8 22:48 my.cnf
user001: drwxr-xr-x 2 opsuser opsuser 4096 Jan 8 22:47 mysql
pdsh>
pdsh> ls -l /mnt
user001: total 66828
user001: -rw-r--r-- 1 root root 68357632 Jan 5 14:06 mysql.tar.gz
pdsh> sudo tar zxvf /mnt/mysql.tar.gz -C /mnt
pdsh> ls -l /mnt
user001: total 66832
user001: drwxr-xr-x 13 root root 4096 Jan 9 2013 mysql
user001: -rw-r--r-- 1 root root 68357632 Jan 5 14:06 mysql.tar.gz
pdsh> sudo rm -rf /mnt/mysql
```

图 6-15 pdsh 的交互模式

交互模式的操作过程和直接执行 pdsh 命令没什么区别，唯一不同的是把在远程执行的命令放到 pdsh 命令行下完成，在 pdsh 命令行模式下，可以依次执行远程操作命令，就像在本地操作一样。这个例子的上半部分是在远程 opsuser 用户下执行创建、复制操作，而下半部分是在远程主机上执行具有 root 权限的文件操作，所以在执行每个命令前，都需要添加 sudo 授权操作。

2. pdcop 应用实例

pdcop 主要用于本地主机与远程主机之间的文件复制，与 pdsh 不同的是，在使用 pdcop 时，本地主机和所有远程主机上都必须安装 pdsh，实际上是因为 pdcop 在进行文件传输时要调用远端主机上的 pdcop 命令，这也是使用 pdcop 比较麻烦的一个方面。不过 pdcop 的使用非常简单，图 6-16 是 pdcop 的两个经典用法。

```
[opsuser@server ~]$ pdcop -R ssh -g userhosts /home/opsuser/mysqlb.tar.gz /home/opsuser
[opsuser@server ~]$ pdcop -R ssh -w userhosts -r /home/opsuser/webdata /home/opsuser
```

图 6-16 pdcop 的用法

在这两个例子中，第一个是将本地文件 /home/opsuser/mysqlb.tar.gz 复制到远程主机组中的 /home/opsuser 目录中，第二个是将本地目录 /home/opsuser/webdata 下的所有文件和子目录都复制到远程主机组中的 /home/opsuser 目录内，这里面用到了“-r”参数，表示递归复制。至此，pdsh 的用法基本介绍完毕了。

6.3 多主机 ssh 封装器 mussh

mussh 的全称是 Multihost SSH Wrapper，它其实是一个 SSH 封装器，由一个 shell 脚本实现。通过 mussh 可以实现批量管理多台远程主机的功能。在大规模自动化运维环境中，mussh 可以起到很好的补充作用。

6.3.1 mussh 功能介绍

mussh 和上面介绍的 pssh、pdsh 类似，都用于批量管理主机，并且可以互为补充，通过 mussh 可以同时多台远程主机上执行 SSH 命令或脚本，这里重点关注 mussh 执行脚本的功能。mussh 可以把本地的脚本在多台远程主机上执行，这个功能十分有用，下面会做重点介绍。

要使用 mussh 的批量管理功能，仍然需要在本地管理主机和远程主机之间做 SSH 的单向信任，这个在介绍 pssh 时已经做过详细介绍，这里不再进行说明。

6.3.2 mussh 的安装与语法

mussh 的安装非常简单，读者可以在 <http://mussh.sourceforge.net/> 获取 mussh 的安装包。目前 mussh 的最新版本是 mussh-1.0。这里下载的软件包为 mussh-1.0.tgz，将安装包解压到本地管理主机上即可完成 mussh 的安装，解压出来的 mussh 命令可直接使用。

执行“`mussh --help`”命令即可获取 `mussh` 的详细用法，这里详细介绍 `mussh` 常用的一些参数及其含义和用法，如表 6-3 所示。

表 6-3 mussh 的常用参数及含义

参数	含义
-m	这个参数用来开启 <code>mussh</code> 的并发执行操作。在管理的主机比较多时，建议设置此参数
-o	此参数可以引入 <code>ssh</code> 参数的具体配置，例如： <code>mussh -o StrictHostKeyChecking=no</code>
-h	指定远程主机，可以指定多个，每个主机之间用空格分隔
-H	指定一组主机列表文件，在管理较多主机的情况下使用
-c	指定在远程主机上要执行的命令，可以执行多个命令，每个命令之间用分号分隔
-C	此参数用来指定一个本地脚本文件，然后此脚本会在多台远程主机上执行
-u	此参数可以去除重复的主机名，是 <code>mussh</code> 的默认值
-U	此参数与“-u”参数相反，不管指定的主机名是否重复
-s	指定在远程主机上执行脚本的 <code>shell</code> 路径，例如： <code>mussh -s /usr/bin/python -C mysql_check.py</code>
-l	指定在远程主机上执行命令的用户

6.3.3 mussh 应用实例

`mussh` 的使用方法与 `pssh`、`pdsh` 基本类似，在完成本地管理主机与远程多台主机之间的 `SSH` 认证之后，就可以使用 `mussh` 批量管理远程服务器了。首先看一个最简单的 `mussh` 应用实例，如图 6-17 所示。

```
[opsuser@server ~]$ mussh -h opsuser@user001 opsuser@user002 -c uptime
opsuser@user001: 01:43:21 up 213 days, 23:13, 5 users, load average: 5.93, 5.74, 5.62
opsuser@user002: 01:43:21 up 213 days, 23:13, 0 users, load average: 8.98, 7.24, 6.12
```

图 6-17 mussh 的应用实例

在这个例子中，主要使用了“-h”和“-c”参数（注意“-h”参数后面多台主机的写法），最后通过“-c”参数指定了在远程主机上要执行的命令，当需要执行多个命令时，每个命令之间用分号分隔。

当需要管理的远程主机较多时，可以将所有主机名或 IP 地址写到一个文件中，然后通过 `mussh` 的“-H”参数调用即可，如图 6-18 所示。

这个例子用到了“-H”和“-l”两个参数，“-H”参数用来指定需要读取的主机列表文件，“-l”参数指定在远程主机上执行命令的用户为 `opsuser`。

如果需要在远程主机上执行的命令较多，一个一个地执行 `mussh` 命令就会变得比较烦琐，此时可以将要执行的所有命令写到一个脚本文件中，然后通过 `mussh` 的“-C”参数调用即可，如图 6-19 所示。


```
[opsuser@server ~]$ more /home/opsuser/myhosts
192.168.16.1
192.168.16.2
192.168.16.3
[opsuser@server ~]$ mussh -H myhosts -l opsuser -c uptime
opsuser@192.168.16.1: 17:18:00 up 214 days, 14:48, 4 users, load average: 5.08, 5.19, 5.45
opsuser@192.168.16.2: 17:18:00 up 214 days, 14:47, 0 users, load average: 5.00, 5.26, 5.44
opsuser@192.168.16.3: 17:18:00 up 208 days, 29 min, 0 users, load average: 6.20, 5.74, 5.75
```

图 6-18 mussh 通过文件读取远程主机列表信息

```
[opsuser@server ~]$ mussh -o "port=22" -H myhosts -s /usr/bin/python -C "/home/opsuser/checkos.py"
192.168.16.1: user001.bestbook.com
192.168.16.2: user002.bestbook.com
192.168.16.2: 192.168.16.2
192.168.16.3: user003.bestbook.com
192.168.16.3: 192.168.16.3
```

图 6-19 通过 mussh 命令将本地脚本在多台远程主机上执行

这个例子用到了“-o”、“-s”和“-C”三个新参数，如果远程主机开放的 ssh 端口不是默认的 22，那么需要通过 mussh 提供的“-o”参数来指定 ssh 的端口，这实际上是通过“-o”参数调用 ssh 服务配置文件 ssh_config 中的参数选项。在执行一个本地脚本时，可以通过“-C”参数实现，这个脚本可以是 shell 脚本、Python 脚本、Perl 脚本等，只要远程服务器上有这个执行环境就行，如果远程服务器上的脚本执行路径不是默认的，那么需要通过“-s”参数指定执行脚本需要的 shell 路径。

至此，mussh 的用法基本介绍完毕了。

7.2 Ganglia 的组成

Ganglia 监控系统由三部分构成，分别是 gmond、gmetad、webfrontend，作用如下。

- gmond: 即 ganglia monitoring daemon，是一个守护进程，运行在每一个需要监测的节点上，用于收集本节点的数据并发送到其他节点，同时也接收其他节点发过来的数据，默认的监听端口为 8642。
- gmetad: 即 ganglia meta daemon，是一个守护进程，运行在一个数据汇聚节点上，定期检查每个监测节点的 gmond 进程并从那里获取数据，然后将数据带标记存储在本地。

分布式监控系统 Ganglia

7.1 Ganglia 简介

Ganglia 是一款为 HPC（高性能计算）集群而设计的可扩展的分布式监控系统，它可以监视和显示集群中节点的各种状态信息，它由运行在各个节点上的 gmond 守护进程来采集 CPU、内存、硬盘利用率、I/O 负载、网络流量情况等方面的数据，然后汇总到 gmetad 守护进程下，使用 rrdtool 存储数据，最后将历史数据以曲线方式通过 PHP 页面呈现。

Ganglia 的特点如下：

- ❑ 良好的扩展性，分层架构设计能够适应大规模服务器集群的需要。
- ❑ 负载开销低，支持高并发。
- ❑ 广泛支持各种操作系统（UNIX 等）和 CPU 架构，支持虚拟机。

7.2 Ganglia 的组成

Ganglia 监控系统由三部分组成，分别是 gmond、gmetad、webfrontend，作用如下。

- ❑ gmond：即 ganglia monitoring daemon，是一个守护进程，运行在每一个需要监测的节点上，用于收集本节点的信息并发送到其他节点，同时也接收其他节点发过来的数据，默认的监听端口为 8649。
- ❑ gmetad：即 ganglia meta daemon，是一个守护进程，运行在一个数据汇聚节点上，定期检查每个监测节点的 gmond 进程并从那里获取数据，然后将数据指标存储在本地

RRD 存储引擎中。

□ webfrontend: 是一个基于 Web 的图形化监控界面，需要和 gmetad 安装在同一个节点上，它从 gmetad 取数据，并且读取 RRD 数据库，通过 rrdtool 生成图表，用于前台展示，界面美观、丰富，功能强大。

图 7-1 是一个简单的 Ganglia 监控系统架构图。

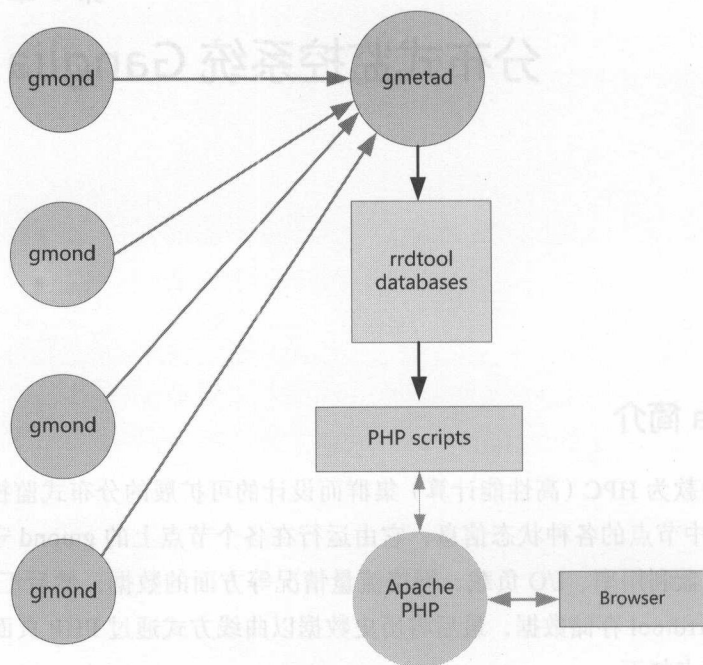


图 7-1 Ganglia 监控系统架构图

从图 7-1 中可以看出，一个 Ganglia 监控系统是由多个 gmond 进程和一个主 gmetad 进程组成，所有 gmond 进程将收集到的监控数据汇总到 gmetad 管理端，而 gmetad 将数据存储到 RRD 数据库中，最后通过 PHP 程序在 Web 界面进行展示。

这是最简单的 Ganglia 运行结构图，在复杂的网络环境下，还有更复杂的 Gnaglia 监控架构。图 7-2 是 Ganglia 的另一种分布式监控架构图。

从图 7-2 中可以看出，gmond 可以等待 gmetad 将监控数据收集走，也可以将监控数据交给其他 gmond，进而让其他 gmond 将数据最终交付给 gmetad，同时，gmetad 也可以收集其他 gmetad 的数据。比如，对于图 7-2 中的 Cluster1 和 Cluster2 集群，Cluster2 就是一个 gmetad，它将自身收集到的数据再次传输给 Cluster1 集群；而 Cluster1 将所有集群的数据进行汇总，然后通过 Web 进行统一展现。

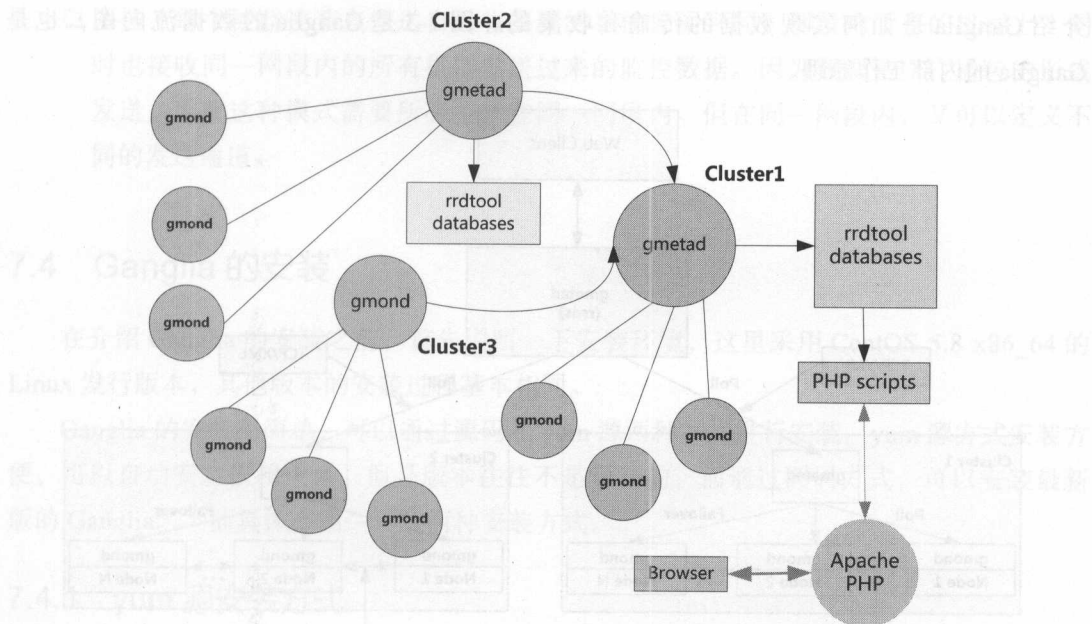


图 7-2 分布式 Ganglia 监控架构图

7.3 Ganglia 的工作原理

在介绍 Ganglia 的工作原理之前，需要介绍一下在 Ganglia 中经常用到的几个名词，这些是了解 Ganglia 分布式架构的基础。在 Ganglia 分布式结构中，经常提到的几个名词有 node、cluster 和 grid，这三部分构成了 Ganglia 分布式监控系统。

- ❑ node: Ganglia 监控系统中的最小单位，即被监控的单台服务器。
- ❑ cluster: 表示一个服务器集群，由多台服务器组成，是具有相同监控属性的一组服务器的集合。
- ❑ grid: 表示一个网格。grid 由多个服务器集群组成，即多个 cluster 组成一个 grid。

从上面的介绍可以看出这三者之间的关系：

- ❑ 一个 grid 对应一个 gmetad，在 gmetad 配置文件中可以指定多个 cluster。
- ❑ 一个 node 对应一个 gmond，gmond 负责采集其所在机器的数据，同时 gmond 还可以接收来自其他 gmond 的数据，而 gmetad 定时去每个 node 上收集监控数据。

7.3.1 Ganglia 数据流向分析

在 Ganglia 分布式监控系统中，gmond 和 gmetad 之间是如何传输数据的呢？接下来

介绍 Ganglia 是如何实现数据的传输和收集的。图 7-3 是 Ganglia 的数据流向图，也是 Ganglia 的内部工作原理。

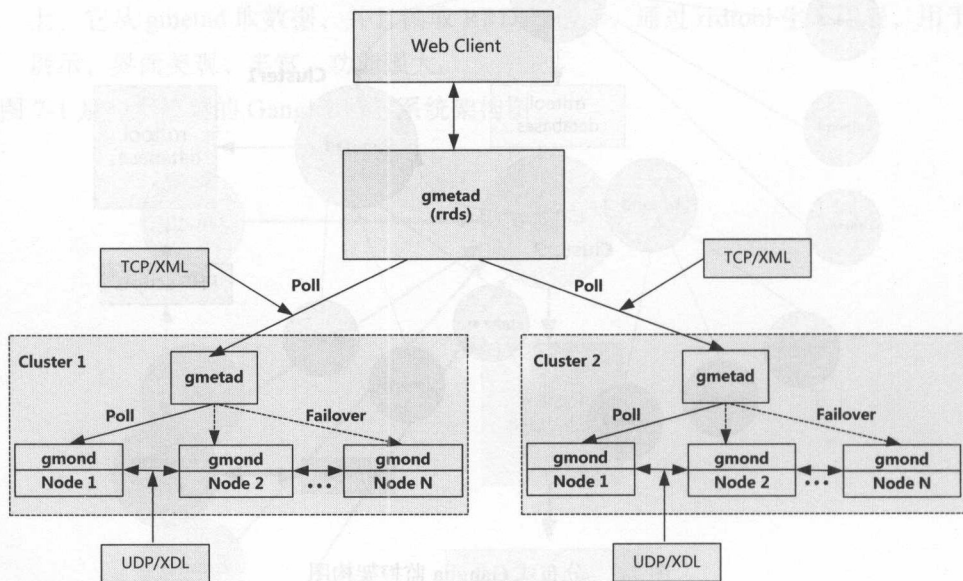


图 7-3 Ganglia 数据流向图

下面简述下 Ganglia 基本运作流程。

- 1) gmond 收集本机的监控数据，发送到其他机器上，并收集其他机器的监控数据，gmond 之间通过 UDP 通信，传递文件格式为 XDL。
- 2) gmond 节点间的数据传输方式支持单播点对点传送外，还支持多播传送。
- 3) gmetad 周期性地到 gmond 节点或 gmetad 节点上获取 (poll) 数据，gmetad 只有 TCP 通道，因此 gmond 与 gmetad 之间的数据都以 XML 格式传输。
- 4) gmetad 既可以从 gmond 也可以从其他的 gmetad 得到 XML 数据。
- 5) gmetad 将获取的数据更新到 rrd 数据库中。
- 6) 通过 Web 监控界面，从 gmetad 取数据，并且读取 rrd 数据库，生成图片显示出来。

7.3.2 Ganglia 工作模式

Ganglia 的收集数据工作可以在单播 (unicast) 或多播 (multicast) 模式下进行，默认为多播模式。

- 单播：每个被监控节点发送自己收集到的本机数据到指定的一台或几台机器上。单播模式可以跨越不同的网段。如果是多个网段的网络环境，就可以采用单播模式采集数据。

- 多播：每个被监控节点发送自己收集到的本机数据到同一网段内所有的机器上，同时也接收同一网段内的所有机器发送过来的监控数据。因为数据是以广播包的形式发送，因此这种模式需要所有主机在同一网段内。但在同一网段内，又可以定义不同的发送通道。

7.4 Ganglia 的安装

在介绍 Ganglia 的安装之前，首先说明一下安装环境，这里采用 CentOS 5.8 x86_64 的 Linux 发行版本，其他版本的安装过程基本相同。

Ganglia 的安装很简单，可以通过源码和 yum 源两种方式进行安装。yum 源方式安装方便，可以自动安装依赖关系，但是版本往往不是最新的，而通过源码方式，可以安装最新版的 Ganglia。下面具体介绍一下这两种安装方式。

7.4.1 yum 源安装方式

CentOS 系统中默认的 yum 源并没有包含 Ganglia，所以我们必须安装扩展的 yum 源。从下面这个地址下载 Linux 附加软件包（EPEL），然后安装扩展 yum 源：

```
[root@node1 ~]# wget http://dl.fedoraproject.org/pub/epel/5/i386/epel-release-5-4.noarch.rpm
[root@node1 ~]# rpm -ivh epel-release-5-4.noarch.rpm
```

完成 yum 源安装，就可以直接通过 yum 源方式安装 Ganglia 了。

Ganglia 的安装分为两个部分，分别是 gmetad 和 gmond，gmetad 安装在监控管理端主机，gmond 安装在需要监控的客户端主机，对应的 yum 包名称分别为 ganglia-gmetad 和 ganglia-gmond。

下面介绍通过 yum 源方式安装 Ganglia 的过程。

以下操作是在监控管理端主机上进行的，首先通过 yum 命令查看下可用的 Ganglia 安装信息：

```
[root@monitor ~]# yum list ganglia*
Available Packages
ganglia.i386                3.0.7-1.el5          epel
ganglia.x86_64              3.0.7-1.el5          epel
ganglia-devel.i386          3.0.7-1.el5          epel
ganglia-devel.x86_64        3.0.7-1.el5          epel
ganglia-gmetad.x86_64       3.0.7-1.el5          epel
ganglia-gmond.x86_64        3.0.7-1.el5          epel
ganglia-web.x86_64          3.0.7-1.el5          epel
```

从输出可知,通过 yum 安装的 Ganglia 版本为 ganglia-3.0.7-1,这个版本并不是最新版本。接着开始安装 ganglia-gmetad:

```
[root@monitor ~]# yum -y install ganglia-gmetad.x86_64
```

安装 gmetad 需要 rrdtool 的支持,而通过 yum 源方式安装,会自动查找 gmetad 依赖的安装包,自动完成安装,这也是 yum 源方式安装的优势。

最后在需要监控的所有客户端主机上安装 gmond 服务:

```
[root@node1 ~]# yum -y install ganglia-gmond.x86_64
```

这样, Ganglia 监控系统就安装完成了。通过 yum 源方式安装的 Ganglia 默认配置文件位于 /etc/ganglia 中。

7.4.2 源码方式

通过源码方式安装 Ganglia 有一定的复杂性,但是可以使用最新的版本,这也是我们推荐的安装方式。源码方式安装 Ganglia 也分为监控管理端的安装和客户端的安装,这里我们安装的是 Ganglia 最新稳定版本 ganglia-3.4.0,安装的路径是 /opt/app/ganglia。首先在监控管理端通过 yum 命令安装 Ganglia 的基础依赖包,操作如下:

```
[root@monitor ~]# yum install -y expat expat-devel pcre pcre-devel zlib cairo-devel  
libxml2-devel pango-devel pango libpng-devel libpng freetype  
freetype-devel libart_lgpl-devel apr-devel
```

接着安装 apr、confuse、rrdtool,操作过程如下:

```
[root@monitor ~]# tar zxvf apr-1.4.6.tar.gz  
[root@monitor ~]# cd apr-1.4.6  
[root@monitor apr-1.4.6]# ./configure  
[root@monitor apr-1.4.6]# make  
[root@monitor apr-1.4.6]# make install  
  
[root@monitor ~]# tar zxvf confuse-2.7.tar.gz  
[root@monitor ~]# cd confuse-2.7  
[root@monitor confuse-2.7]# ./configure CFLAGS=-fPIC --disable-nls  
[root@monitor confuse-2.7]# make  
[root@monitor confuse-2.7]# make install  
  
[root@monitor ~]# tar zxvf rrdtool-1.4.7.tar.gz  
[root@monitor ~]# cd rrdtool-1.4.7  
[root@monitor rrdtool-1.4.7]# ./configure --disable-tcl --prefix=/opt/rrdtool  
[root@monitor rrdtool-1.4.7]# make  
[root@monitor rrdtool-1.4.7]# make install
```

最后安装 ganglia-gmetad,过程如下:

```

[root@monitor ~]# tar zxvf ganglia-3.4.0.tar.gz
[root@monitor ~]# cd ganglia-3.4.0
[root@monitor ganglia-3.4.0]# ./configure --prefix=/opt/app/ganglia \
>--with-static-modules --enable-gexec --enable-status --with-gmetad --with-python=/usr \
>--with-librrd=/opt/rrdtool --with-libexpat=/usr --with-libconfuse=/usr/local \
>--with-libpcrc=/usr/local
[root@monitor ganglia-3.4.0]# make;make install
[root@monitor ganglia-3.4.0]# cd gmetad
[root@monitorgmetad]# cp gmetad.conf /opt/app/ganglia/etc/# 复制 gmetad 服务配置文件
[root@monitorgmetad]# cp gmetad.init /etc/init.d/gmetad # 复制 gmetad 服务启动脚本
# 到 /etc/init.d 目录
[root@monitorgmetad]# sed -i \
>"s/^GMETAD=\/usr\/sbin\/gmetad/GMETAD=\/opt\/app\/ganglia\/sbin\/gmetad/g" \
>/etc/init.d/gmetad # 修改 /etc/init.d/gmetad 文件中 gmetad 命令的默认路径
[root@monitorgmetad]# chkconfig --add gmetad
[root@monitorgmetad]# ip route add 239.2.11.71 dev eth0

```

需要说明的一点是：239.2.11.71 这个地址是 Ganglia 默认的多播地址，将这个多播地址绑定到相应的网卡设备（这里是 eth0，可根据不同的环境进行修改）上即可。设置了多播地址后，Ganglia 管理端就可以发送和接收多播信息。

至此，ganglia-gmetad 安装完成。

下面介绍 Ganglia 客户端的安装过程，ganglig-gmond 的安装与 ganglia-gmetad 大致相同，对于系统依赖包和基础软件包的安装完全相同，只是 ganglig-gmond 不需要 rrdtool 的支持，因此重点讲述 ganglig-gmond 的编译安装过程。

```

[root@node1 ~]# tar zxvf ganglia-3.4.0.tar.gz
[root@node1 ~]# cd ganglia-3.4.0
[root@node1 ganglia-3.4.0]# ./configure --prefix=/opt/app/ganglia --enable-gexec \
>--enable-status --with-python=/usr --with-libapr=/usr/local/apr/bin/apr-1-config \
>--with-libconfuse=/usr/local --with-libexpat=/usr --with-libpcrc=/usr
[root@node1 ganglia-3.4.0]# make
[root@node1 ganglia-3.4.0]# make install
[root@node1 gmond]# cd gmond
[root@node1 gmond]# ./gmond -t > /opt/app/ganglia/etc/gmond.conf
# 用于生成 gmond 服务配置文件
[root@node1 gmond]# cp gmond.init /etc/init.d/gmond
# 复制 gmond 服务启动脚本到 /etc/init.d 目录
[root@node1 gmond]# sed -i \
>"s/^GMOND=\/usr\/sbin\/gmond/GMOND=\/opt\/app\/ganglia\/sbin\/gmond/g" \
>/etc/init.d/gmond # 修改 /etc/init.d/gmond 文件中 gmond 命令的默认路径
[root@node1 gmond]# chkconfig --add gmond
[root@node1 gmond]# ip route add 239.2.11.71 dev eth0

```

与 ganglia-gmetad 的安装相同，在 ganglig-gmond 中也需要把 239.2.11.71 这个广播地

址绑定到系统对应网卡上。

到这里为止，ganglig-gmond 安装完成。

7.5 配置一个 Ganglia 分布式监控系统

7.5.1 Ganglia 配置文件介绍

Ganglia 的配置文件主要有两个，分别是监控管理端的 gmetad.conf 和客户端的 gmond.conf 文件。根据 Ganglia 安装方式的不同，配置文件的路径也不相同，通过 yum 源方式安装的 Ganglia，默认的配置文件的位于 /etc/ganglia 下，而通过源码方式安装的 Ganglia，配置文件路径位于 Ganglia 安装路径的 etc 目录下，例如，前面通过源码方式安装的 Ganglia 配置文件路径为 /opt/app/ganglia/etc。在监控管理端，只需要配置 gmetad.conf 文件即可，而在客户端也只需要配置 gmond.conf 文件就行了。

7.5.2 Ganglia 监控系统架构图

Ganglia 支持多种监控架构，这是由 gmetad 的特性决定的，gmetad 可以周期性地多个 gmond 节点收集数据，这就是 ganglia 的两层架构。同时，gmetad 不但可以从 gmond 收集数据，也可以从其他的 gmetad 得到数据，这就形成了 Gnanglia 的三层架构。多种架构方式也体现了 Ganglia 作为分布式监控系统的灵活性和扩展性。

这里介绍一个简单的 Ganglia 配置架构，即一个监控管理端和多个客户端的两层架构。我们假定 gmond 工作在多播模式，并且有一个 Cluster1 集群，其中有 4 台要监控的服务器，主机名从 cloud0 到 cloud3，这 4 台主机在同一个网段内。

7.5.3 Ganglia 监控管理端配置

监控管理端的配置文件是 gmetad.conf，这个配置文件内容比较多，但是需要修改的配置仅有如下几个：

```
data_source "Cluster1" cloud0 cloud2
gridname "TopGrid"
xml_port 8651
interactive_port 8652
rrd_rootdir "/var/lib/ganglia/rrds"
```

各参数含义如下。

□ data_source：此参数定义了集群名字，以及集群中的节点。Cluster1 就是这个集群的

名称, cloud0 和 cloud2 指明了从这两个节点收集数据, Cluster1 后面指定的节点名可以是 IP 地址, 也可以是主机名, 由于采用了多播模式, 每个 gmond 节点都有本 Cluster1 集群节点的所有监控数据, 因此不需要把所有节点的都写入 data_source 中。但是建议写入不低于 2 个, 这样, 在 cloud0 节点出现故障的时候, gmetad 会自动到 cloud2 节点采集数据, 这样就保证了 Ganglia 监控系统的高可用性。

上面通过 data_source 参数定义了一个服务器集群 Cluster1, 对于要监控多个应用系统的情况, 还可以对不同用途的主机进行分组, 定义多个服务器集群, 分组方式可以通过下面的方法定义:

```
data_source "my cluster" 10 localhost my.machine.edu:8649 1.2.3.5:8655
data_source "my grid" 50 1.3.4.7:8655 grid.org:8651 grid-backup.org:8651
data_source "another source" 1.3.4.7:8655 1.3.4.8
```

可以通过定义多个 data_source 来实现监控多个服务器集群, 而每个服务器集群在定义集群节点的时候, 可以采用主机名或 IP 地址等形式, 也可以加端口, 如果不加端口, 默认端口是 8649, 同时可以设定采集数据的频率, 如上面的“10 localhost、50 1.3.4.7:8655”等, 分别表示每隔 10s、50s 采集一次数据。

❑ gridname: 此参数是定义一个网格名称。一个网格有多个服务器集群组成, 每个服务器集群由“data_source”选项来定义。

❑ xml_port: 此参数定义了一个收集数据汇总的交互端口, 如果不指定, 默认是 8651, 可以通过 telnet 这个端口得到监控管理端收集到的客户端的所有数据。

❑ interactive_port: 此参数定义了 Web 端获取数据的端口, 这个端口在配置 Ganglia 的 Web 监控界面时需要指定。

❑ rrd_rootdir: 此参数定义了 rrd 数据库的存放路径, gmetad 在收集到监控数据后会将其更新到该目录下对应的 rrd 数据库中。

到这里为止, 在 Ganglia 监控管理端的配置完成了。

7.5.4 Ganglia 的客户端配置

Ganglia 监控的客户端 gmond 安装完成后, 配置文件位于 Ganglia 安装路径的 etc 目录下, 名称为 gmond.conf, 这个配置文件稍微复杂, 如下所示:

```
globals {
daemonize = yes      # 是否后台运行, 这里表示以后台的方式运行
setuid = yes         # 是否设置运行用户, 在 Windows 中需要设置为 false
user = nobody        # 设置运行的用户名称, 必须是操作系统已经存在的用户, 默认是 nobody
debug_level = 0      # 调试级别, 默认是 0, 表示不输出任何日志, 数字越大表示输出的日志越多
max_udp_msg_len = 1472
```

```

mute = no          # 是否发送监控数据到其他节点, 设置为“no”表示本节点将不再广播任何自己收集
                  # 到的数据到网络上
deaf = no          # 是否接受其他节点发送过来的监控数据, 设置为“no”表示本节点将不再接收任何
                  # 其他节点广播的数据包
allow_extra_data = yes # 是否发送扩展数据
host_dmax = 0 /*secs */ # 是否删除一个节点, 0 代表永远不删除, 0 之外的整数代表节点的不响应时间,
                  # 超过这个时间后, Ganglia 就会刷新集群节点信息进而删除此节点
cleanup_threshold = 300 /*secs */ # gmond 清理过期数据的时间
gexec = no        # 是否使用 gexec 来告知主机是否可用, 这里不启用
send_metadata_interval = 0 # 在单播协议中, 新添加的节点在多长时间以内响应一下以表示自己的存在,
                  # 0 代表仅在 gmond 启动时通知一次, 单位为 s
}
cluster {
name = "Cluster1"      # 集群的名称, 是区分此节点属于某个集群的标志, 必须和监控服务端
                  # data_source 中的某一项名称匹配
owner = "junfeng"      # 节点的拥有者, 也就是节点的管理员
latlong = "unspecified" # 节点的坐标, 如经度、纬度等, 一般无需指定
url = "unspecified"    # 节点的 URL 地址, 一般无需指定
}
host {
location = "unspecified" # 节点的物理位置, 一般无需指定
}
udp_send_channel {
mcast_join = 239.2.11.71 # 指定发送的多播地址, 其中 239.2.11.71 是一个 D 类地址。如果使用
                        # 单播模式, 则要写 host = host1, 在单播模式下也可以配置多个
                        # udp_send_channel
port = 8649             # 监听端口
ttl = 1
}
udp_rcv_channel {
mcast_join = 239.2.11.71 # 指定接收的多播地址, 同样也是 239.2.11.71 这个 D 类地址
port = 8649             # 监听端口
bind = 239.2.11.71      # 绑定地址
}
tcp_accept_channel {
port = 8649             # 通过 TCP 协议监听的端口, 在远端可以通过连接到 8649 端口得到监控数据
}

```

在一个集群内, 所有客户端的配置是一样的。完成一个客户端配置后, 将配置文件复制到此集群内的所有客户端主机上即可完成客户端主机的配置。

7.5.5 Ganglia 的 Web 端配置

Ganglia 的 Web 监控界面是基于 PHP 的, 因此需要安装 PHP 环境。PHP 环境的安装这里不做介绍, 读者可以在 <http://sourceforge.net/projects/ganglia/files/> 下载 ganglia-web 的最

新版本, 然后将 ganglia-web 程序放到 Apache Web 的根目录即可, 这里我们推荐下载的版本是 ganglia-web-3.5.7。

配置 Ganglia 的 Web 界面比较简单, 只需要修改几个 PHP 文件即可。首先是 conf_default.php, 可以将 conf_default.php 重命名为 conf.php, 也可以保持不变, Ganglia 的 Web 默认先找 conf.php, 找不到会继续找 conf_default.php, 需要修改的内容如下:

```
$conf['gweb_confdir'] = "/var/www/html/ganglia"; # ganglia web 的根目录
$conf['gmetad_root'] = "/opt/app/ganglia"; # ganglia 程序安装目录
$conf['rrds'] = "${conf['gmetad_root']}/rrds"; # ganglia web 读取 rrd 数据库的路径,
这里是 /opt/app/ganglia/rrds
$conf['dwoo_compiled_dir'] = "${conf['gweb_confdir']}/dwoo/compiled"; # 需要“777”权限
$conf['dwoo_cache_dir'] = "${conf['gweb_confdir']}/dwoo/cache"; # 需要“777”权限
$conf['rrdtool'] = "/opt/rrdtool/bin/rrdtool"; # 指定 rrdtool 的路径
$conf['graphdir'] = $conf['gweb_root'] . '/graph.d'; # 生成图形模板目录
$conf['ganglia_ip'] = "127.0.0.1"; # gmetad 服务所在服务器的地址
$conf['ganglia_port'] = 8652; # gmetad 服务器的交互式提供监控数据端口发布
```

这里需要说明的是: “\$conf['dwoo_compiled_dir']” 和 “\$conf['dwoo_cache_dir']” 指定的路径在默认情况下可能不存在, 因此需要手动建立 compiled 和 cache 目录, 并授予 Linux 下“777”的权限。另外, rrd 数据库的存储目录 /opt/app/ganglia/rrds 一定要保证 rrdtool 可写, 因此需要执行授权命令:

```
chown -R nobody:nobody /opt/app/ganglia/rrds
```

这样 rrdtool 才能正常读取 rrd 数据库, 进而将数据通过 Web 界面展示出来。其实 ganglia-web 的配置还是比较简单的, 一旦配置出错会给出提示, 根据错误提示进行问题排查, 一般都能找到解决方法。

7.6 Ganglia 监控系统的管理和维护

在 Ganglia 的所有配置完成之后, 就可以启动 Ganglia 监控服务了, 首先在被监控节点依次启动 gmond 服务, 操作如下:

```
[root@node1 ~]# /etc/init.d/gmond start
```

然后通过查看系统的 /var/log/messages 日志信息, 判断 gmond 是否成功启动, 如果出现问题, 根据日志的提示进行解决。

接着就可以启动监控管理节点的 gmetad 服务了, 操作如下:

```
[root@monitor ~]# /etc/init.d/gmetad start
```


同样，也可以跟踪一下系统的 /var/log/messages 日志信息，看启动过程是否出现异常。

最后，将 Apache/PHP 的 Web 服务启动，就可以查看 Ganglia 收集到的所有节点的监控数据信息。图 7-4 是 Ganglia Web 某一时刻的运行状态图。

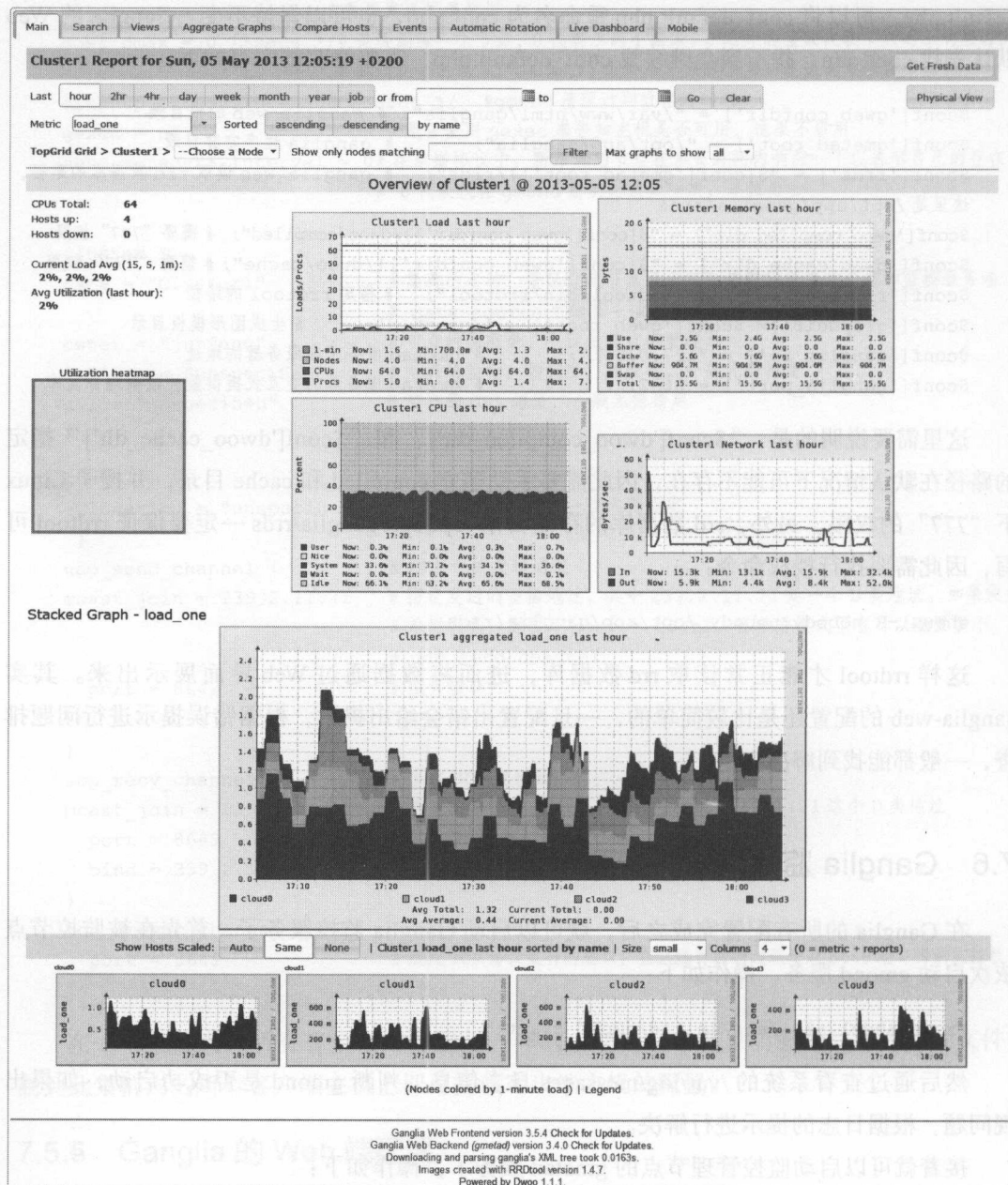


图 7-4 Ganglia Web 某一时刻的运行状态图

7.7 Ganglia 监控扩展实现机制

在默认情况下, Ganglia 通过 gmond 守护进程收集 CPU、内存、磁盘、IO、进程、网络 6 大方面的数据, 然后汇总到 gmetad 守护进程下, 使用 rrdtools 存储数据, 最后将历史数据以曲线方式通过 PHP 页面呈现。但是很多时候这些基础数据还不足以满足我们的监控需要, 我们还要根据应用的不同, 扩展 Ganglia 的监控范围。下面我们就介绍通过开发 Ganglia 插件扩展 Ganglia 监控功能的实现方法。

7.7.1 扩展 Ganglia 监控功能的方法

默认安装完成的 Ganglia 仅向我们提供基础的系统监控信息, 通过 Ganglia 插件可以实现两种扩展 Ganglia 监控功能的方法。

1) 添加带内 (in-band) 插件, 主要是通过 gmetric 命令来实现。

这是通常使用的一种方法, 主要是通过 crontab 方法并调用 Ganglia 的 gmetric 命令来向 gmond 输入数据, 进而实现统一监控。这种方法简单, 对于少量的监控可以采用, 但是对于大规模自定义监控时, 监控数据难以统一管理。

2) 添加一些其他来源的带外 (out-of-band) 插件, 主要是通过 C 或者 Python 接口来实现。

在 Ganglia3.1.x 版本以后, 增加了 C 或 Python 接口, 通过这个接口可以自定义数据收集模块, 并且可以将这些模块直接插入 gmond 中以监控用户自定义的应用。

7.7.2 通过 gmetric 接口扩展 Ganglia 监控

gmetric 是 Ganglia 的一个命令行工具, 它可以将数据直接发送到负责收集数据的 gmond 节点, 或者广播给所有 gmond 节点。由此可见, 采集数据的不一定全部都是 gmond 这个服务, 也可以直接通过应用程序调用 Ganglia 提供的 gmetric 工具将数据直接写入 gmond 中, 这就很容易地实现了 ganglia 监控的扩展。因此, 我们可以利用 shell、Perl、Python 等语言工具, 通过调用 gmetric 将我们想要监控的数据直接写入 gmond 中, 简单而快速地实现了 Ganglia 的监控扩展。

在 Ganglia 安装完成后, 会在 bin 目录下生成 gmetric 命令。下面通过一个实例介绍一下 gmetric 的使用方法:

```
[root@cloud1 ~]# /opt/app/ganglia/bin/gmetric \
>-n disk_used -v 40 -t int32 -u '% test' -d 50 -S '8.8.8.8:cloud1'
```

其中:

□ -n, 表示要监控的指标名。

- ❑ -v, 表示写入的监控指标值。
- ❑ -t, 表示写入监控数据的类型。
- ❑ -u, 表示监控数据的单位。
- ❑ -d, 表示监控指标的存活时间。
- ❑ -c, 用于指定 Ganglia 配置文件的位置。
- ❑ -S, 表示伪装客户端信息, 8.8.8.8 代表伪装的客户端地址, cloud1 代表被监控主机的主机名。

通过不断地执行 gmetric 命令写入数据, Ganglia Web 的监控报表已经形成, 如图 7-5 所示。

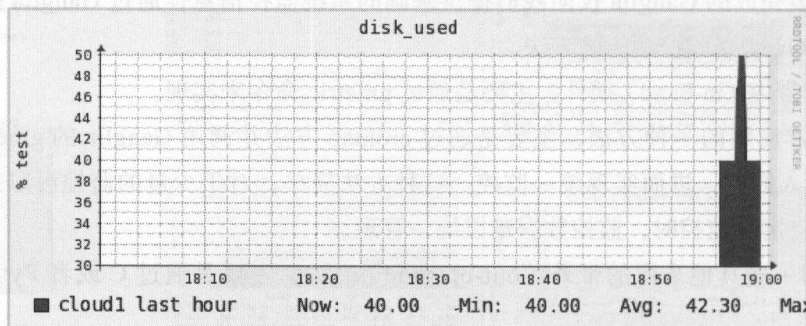


图 7-5 通过 gmetric 生成的监控报表

从图 7-5 中可以看到刚才执行命令时设置的几个属性值在报表中都呈现出来了, 例如 disk_used、“% test”、cloud1 等。同时, 通过 gmetric 写入的监控数值, 在报表中也很清楚地展示出来。

在上面的实例中, 我们通过执行命令的方式不断写入数据, 进而生成监控报表。事实上, 所有的监控数据都是自动收集的, 因此, 要实现数据的自动收集, 可以将上面的命令写成一个 shell 脚本, 然后将脚本文件放入 cron 运行。

假设生成的脚本文件是 /opt/ganglia/bin/ganglia.sh, 运行 “crontab -e” 命令, 将此脚本每隔 10 分钟运行一次:

```
*/10 * * * * /opt/ganglia/bin/ganglia.sh
```

最后, 打开 Ganglia Web 进行浏览, 即可看到通过 gmetric 命令收集到的数据报表。

7.7.3 通过 Python 插件扩展 Ganglia 监控

要通过 Python 插件扩展 Ganglia 监控, 必须满足如下条件:

- ❑ Ganglia 3.1.x 以后版本

□ Python2.6.6 或更高版本

□ Python 开发头文件（通常在 python-devel 这个软件包中）

在安装 Ganglia 客户端（gmond）的时候，需要加上“--with-python”参数，这样在安装完成后，会生成 modpython.so 文件，这个文件是 Ganglia 调用 Python 的动态链接库，要通过 Python 接口开发 Ganglia 插件，必须编译安装此 Python 模块。

这里假定 Ganglia 的安装版本是 ganglia3.4.0，安装目录是 /opt/app/ganglia，要编写一个基于 Python 的 Ganglia 插件，需要进行如下操作。

1. 修改 modpython.conf 文件（Ganglia 客户端）

在 Ganglia 安装完成后，modpython.conf 文件位于 /opt/app/ganglia/etc/conf.d 目录下，此文件内容如下：

```
modules {
    module {
        name = "python_module"           # python 主模块名称
        path = "modpython.so"
        #Ganglia 调用 Python 的动态链接库，这个文件应该在 Ganglia 的安装目录的 lib64/ganglia 下面
        params = "/opt/app/ganglia/lib64/ganglia" # 指定我们编写的 Python 脚本放置路径，这个
        # 路径要保证是已存在的。否则 gmond 服务无法启动
    }
}
include ("/opt/app/ganglia/etc/conf.d/*.pyconf") # Python 脚本配置文件存放路径
```

2. 重启 gmond 服务

在客户端的所有配置修改完成后，重启 gmond 服务即可完成 Python 接口环境的搭建。

7.7.4 实战：利用 Python 接口监控 Nginx 运行状态

搭建完 Python 接口环境，这只是实现 Ganglia 监控扩展的第一步，接下来还要编写基于 Python 的 Ganglia 监控插件。不过幸运的是，网上有很多已经编写好的各种应用服务的监控插件，我们只需要拿来使用即可。读者可以从 https://github.com/ganglia/gmond_python_modules 下载需要的各种 Ganglia 扩展监控插件，这里要下载的是 nginx_status 这个 Python 插件。下载完成的 nginx_status 插件的目录结构如下：

```
[root@cloudlinux_status]# ls
conf.d graph.d python_modules README.mkdn
```

其中，conf.d 目录下存放的是配置文件 nginx_status.pyconf，python_modules 目录下放的是 Python 插件的主程序 nginx_status.py，graph.d 目录下放的是用于绘图的 PHP 程序。这几个文件接下来都会用到。

对 Nginx 的监控，需要借助 with-http_stub_status_module 模块，此模块默认不开启，所以需要指定开启，用于编译 Nginx。关于安装与编译 Nginx，这里不进行介绍。

1. 配置 Nginx，开启状态监控

在 Nginx 配置文件 nginx.conf 中添加如下配置：

```
server {
    listen 8000;           # 监听的端口
    server_name IP;        # 当前机器的 IP 地址或域名
    location /nginx_status {
        stub_status on;
        access_log off;
        # allow xx.xx.xx.xx; # 允许访问的 IP 地址
        # deny all;
    }
    allow all;
}
```

接着，重启 Nginx，通过 http://IP:8000/nginx_status 即可看到状态监控结果。

2. 配置 Ganglia 客户端，收集 nginx_status 数据

根据前面对 modpython.conf 文件的配置，我们将 nginx_status.pyconf 文件放到 /opt/app/ganglia/etc/conf.d 目录下，将 nginx_status.py 文件放到 /opt/app/ganglia/lib64/ganglia 目录下。

nginx_status.py 文件无需改动，nginx_status.pyconf 文件需要做一些修改，修改后的文件内容如下：

```
[root@cloud1 conf.d]# morenginx_status.pyconf

modules {
    module {
        name = 'nginx_status' # 模块名，该文件存放于 /opt/app/ganglia/lib64/ganglia 下面
        language = 'python'   # 声明使用 Python 语言
    }
    paramstatus_url {
        value = 'http://IP:8000/nginx_status' # 这个就是查看 nginx 状态的 URL 地址，前面
                                                # 有配置说明
    }
    paramnginx_bin {
        value = '/usr/local/nginx/sbin/nginx' # 这里假定 Nginx 安装路径为 /usr/local/nginx
    }
    paramrefresh_rate {
        value = '15'
    }
}
```

```

    }
}
# 下面是需要收集的 metric 列表，一个模块中可以扩展任意个 metric
collection_group {
    collect_once = yes
    time_threshold = 20
    metric {
        name = 'nginx_server_version'
        title = "Nginx Version"
    }
}
collection_group {
    collect_every = 10
    time_threshold = 20
    metric {
        name = "nginx_active_connections" # 最大发送间隔
        title = "Total Active Connections" # metric 在模块中的名字
        value_threshold = 1.0 # 图形界面上显示的标题
    }
    metric {
        name = "nginx_accepts"
        title = "Total Connections Accepted"
        value_threshold = 1.0
    }
    metric {
        name = "nginx_handled"
        title = "Total Connections Handled"
        value_threshold = 1.0
    }
    metric {
        name = "nginx_requests"
        title = "Total Requests"
        value_threshold = 1.0
    }
    metric {
        name = "nginx_reading"
        title = "Connections Reading"
        value_threshold = 1.0
    }
    metric {
        name = "nginx_writing"
        title = "Connections Writing"
        value_threshold = 1.0
    }
    metric {

```

```

name = "nginx_waiting"
title = "Connections Waiting"
value_threshold = 1.0
}
}

```

3. 绘图展示的 PHP 文件

在完成数据收集后，还需要将数据以图表的形式展示在 Ganglia Web 界面中，所以还需要前台展示文件，将 graph.d 目录下的两个文件 nginx_accepts_ratio_report.php 和 nginx_scoreboard_report.php 放到 Ganglia Web 的绘图模板目录即可。根据上面的设定，Ganglia Web 的安装目录是 /var/www/html/ganglia，因此，将上面这两个 PHP 文件放到 /var/www/html/ganglia/graph.d 目录下即可。

4. nginx_status.py 输出效果图

完成前面所有步骤后，重启 Ganglia 客户端 gmond 服务，在客户端通过“gmond -m”命令可以查看支持的模板，最后就可以在 Ganglia Web 界面查看 Nginx 的运行状态，如图 7-6 所示。

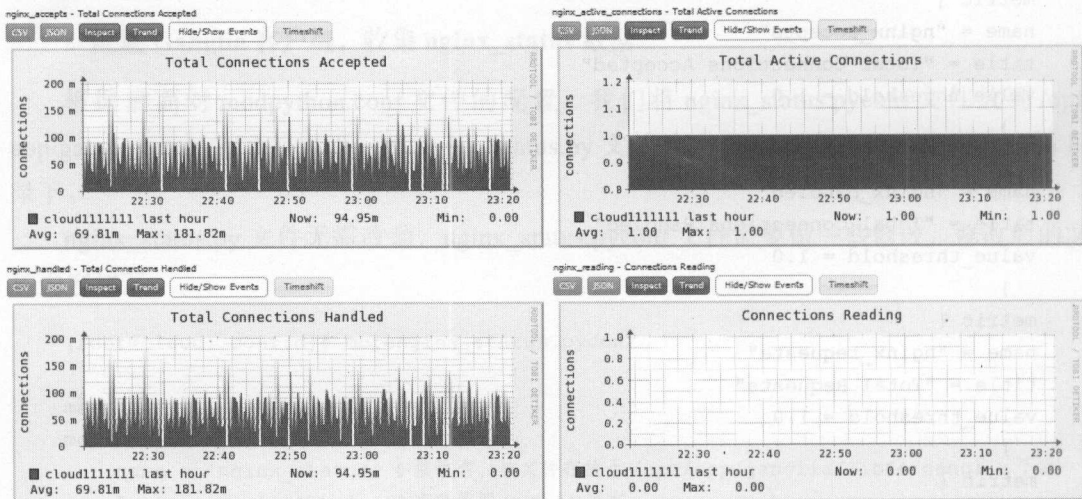


图 7-6 通过 Python 插件实现 Ganglia 监控 Nginx

7.8 Ganglia 在实际应用中要考虑的问题

7.8.1 网络 IO 可能存在瓶颈

在 Ganglia 分布式监控系统中，运行在被监控节点上的 gmond 进程消耗的网络资源是

非常小的, 通常在 1~2MB 之间, 而 gmond 将收集到的数据仅保存在内存中, 因此 gmond 消耗的网络资源基本可以忽略不计。但有一种情况, 就是在采用单播模式下, 所有 gmond 进程都会向一个 gmond 中央节点发送数据, 而这个 gmond 中央节点可能存在网络开销, 如果单播传输的节点过多, 那么在中央节点上就会存在网络 IO 瓶颈。

另外, gmetad 管理节点会收集所有 gmond 节点上的监控数据, 同时 Ganglia-Web 也运行在 gmetad 所在的节点上, 因此, gmetad 所在节点的网络 IO 也会很大, 可能存在网络 IO 瓶颈。

7.8.2 CPU 可能存在瓶颈

对于 gmetad 管理节点, 它将收集所有 gmond 节点收集到的 UDP 数据包, 如果一个节点每秒发送 10 个数据包, 300 个节点每秒将会发送 3000 个, 假如每个数据包 300B, 那么每秒就有近 1MB 的数据, 这么多数据包需要的 CPU 处理能力也会增加。

gmetad 在默认情况下每 15s 到 gmond 取一次数据, 同时 gmetad 请求完数据后还要汇总到 XML 文件, 还需要对 XML 文件进行解析, 如果监控的节点较多, 比如 1000 个节点, 那么收集到的 XML 文件可能有 10~20MB 左右。如果按照默认情况每隔 15s 去解析一个 20MB 左右的 XML 文件, 那么 CPU 将面临很大压力, 而 gmetad 还要将数据写入 rrd 数据库, 同时还要处理来自 Web 客户端的解析请求进而读 rrd 数据库, 这些都会加重 CPU 的负载, 因此在监控的节点比较多时, gmetad 节点应该选取性能比较好的服务器, 特别是 CPU 性能。

7.8.3 gmetad 写入 rrd 数据库可能存在瓶颈

gmetad 进程在收集完成客户端的监控数据后, 会通过 rrdtool 工具将数据写入到 rrd 数据库的存储目录。由于 rrd 拥有独特的存储方式, 它将每个 metric 作为一个文件来存储, 并且如果配置了数据采集的频率, gmetad 还会为每个采集频率保存一个单独的文件, 这就意味着, gmetad 将 metric 值保存到 rrd 数据库的操作将是针对大量小文件的 IO 操作。假设集群有 500 个节点, 每个节点有 50 个 metric, 那么 gmetad 将会存储 25000 个 metric, 如果这些 metric 都是每秒更新一次, 这将意味着每秒有 25000 个随机写入操作, 而对于这种写入操作, 一般的硬盘是无法支撑的。

基于 nagios 的分布式监控报警平台 Centreon

8.1 Centreon 概述

Centreon 是一款功能强大的分布式 IT 监控系统，它通过第三方组件可以实现对网络、操作系统和应用程序的监控：首先，它是开源的，我们可以免费使用它；其次，它的底层采用 nagios 作为监控软件，同时 nagios 通过 ndoutil 模块将监控到的数据定时写入数据库中，而 Centreon 实时从数据库读取该数据并通过 Web 界面展现监控数据；最后，我们可以通过 Centreon 管理和配置 nagios，或者说 Centreon 就是 nagios 的一个管理配置工具，通过 Centreon 提供的 Web 配置界面，可以轻松完成 nagios 的各种烦琐配置。

此外，Centreon 还支持 NRPE、SNMP、NSClient 等插件，可以通过这些插件构建分布式监控报警系统。

8.2 Centreon 的特点

Centreon 作为一款优秀的 IT 监控软件，具有如下显著特点和优点：

- ❑ 采用 Web 方式配置 nagios，通过 Web 界面即可完全控制 nagios，可以轻松添加和管理上千台主机和服务。
- ❑ 支持主机模板和服务模板，并且自动建立关联服务。
- ❑ 支持多节点的分布式监控，对于远程节点可采用 NRPE、SNMP、NSClient 等方式监控私有服务。

- ❑ 支持 ACL 权限管理，可以给不同用户设置不同的管理权限，多个用户可以管理不同主机和服务，互不影响。
- ❑ 详细的报表统计功能和日志管理功能，可以查看某个时间段某个服务或主机的运行状态、故障率等。
- ❑ 模块化管理，可根据需要定制自己的模块，同时支持第三方监控数据接入。

8.3 Centreon 的结构

一个典型的 Centreon 监控系统一般由四大部分组成，分别是 nagios、centstorage、centcore 和 ndoutils，简单介绍如下：

- ❑ nagios 是 Centreon 的底层监控引擎，主要完成监控报警系统所需的各项功能，是 Centreon 监控系统的核心。另外，Centreon 还支持 Centreon Engine、Icinga 等监控引擎。本章采用 nagios 监控引擎进行介绍。
- ❑ centstorage 是一个数据存储模块，它主要用于将日志数据及 RRDtool 生成的数据存储在数据库中，以供用户查询日志数据并快速生成曲线图，更主要的是 nagios 可以随时通过查看数据库中的记录更新监控状态。
- ❑ centcore 主要用于 centreon 的分布式监控系统中，在系统中 centcore 是一个基于 Perl 的守护进程，主要负责中心服务器（central server）和扩展节点（poller）间的通信和数据同步等操作，例如 centcore 可以在中心服务器上执行对远程扩展节点上 nagios 服务的启动、关闭和重启操作，还可以运行、更新扩展节点上 nagios 的配置文件。
- ❑ ndoutils 是将 nagios 与数据库进行连接的工具，它可以将 nagios 的实时状态写入数据库，以供其他程序调用，最终可以实现在一个控制台上完成所有扩展节点的数据入库操作。

在介绍了 Centreon 监控系统的基本组成以后，接下来重点看看每个组成部分是如何协调工作的，如图 8-1 所示。

图 8-1 主要展示了 Centreon 每个组成部分是如何工作的，在组织结构上，一般情况下，Centreon Web、Centstorage、Centcore 和 Ndo2DB 位于中心服务器上，而 nagios 和 ndomod 可以位于一台独立的扩展节点（poller）上，也可以位于中心服务器上。在分布式监控环境中，nagios 和 ndomod 都位于远程的一个扩展节点上，此图没有展示 Centreon 的分布式监控架构，这个将在 8.6 节详细介绍。

为了能使读者快速了解 Centreon 的内部工作原理，我们将图 8-1 分为三条线来介绍，第一条线：Centreon Web → Centcore → Centstorage → db → Centreon Web，Centreon Web 就是 Centreon 的 Web 配置管理界面，在 Web 配置管理界面中配置好主机和服务后，会生

成相应的配置文件，然后 Centcore 会去读取这些配置文件并结合相关 nagios 插件将数据发送到 nagios 监控引擎，并生成相关日志文件和 rrd 文件，而 Centstorage 模块会及时收集这些日志信息及 rrd 数据并最终将这些数据存入数据库中，以供 Centreon Web 展示调用。

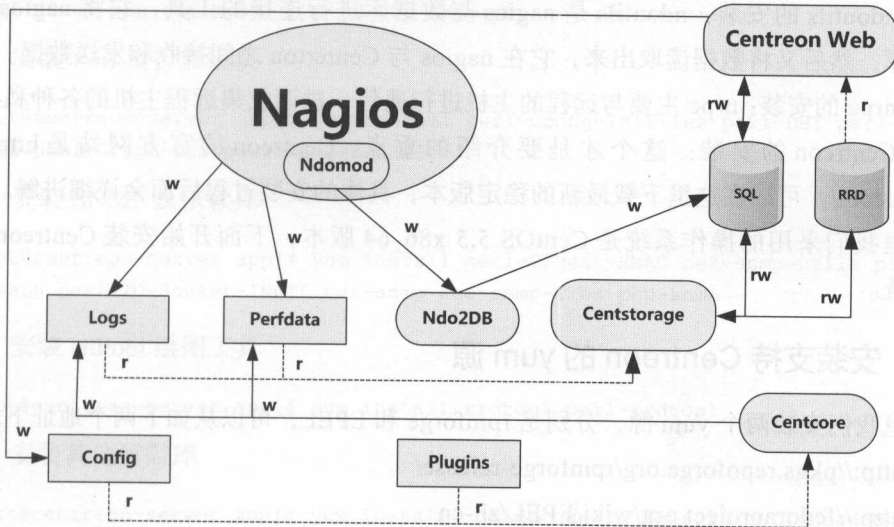


图 8-1 典型的 Centreon 监控系统的内部工作原理

第二条线：nagios → Centstorage → db → Centreon Web，在本地或远程的扩展节点中 nagios 监控引擎会产生日志文件和 rrd 文件，这些文件定期被 Centstorage 读取并最终存储在数据库中，以供 Centreon Web 读取。

第三条线：nagios (ndomod) → Ndo2DB → db → Centreon Web，这一条线将 nagios 实时监控状态写入数据库，首先由在本地或远程扩展节点上的 ndomod 进程将 nagios 监控状态通过 Ndo2DB 模块写入数据库，最后 Centreon Web 会定期调用此 db 库，这样监控系统中各个主机或服务的监控状态就被实时展示出来了。

通过对这三条线的描述，Centreon 内部工作过程就变得非常清晰了！

8.4 安装 Centreon+nagios 监控系统

Centreon 的安装有一定的复杂性，对操作系统库依赖较多，在安装方式上有源码编译安装和 yum 源安装两种，源码编译安装较复杂，出错几率也较高，因此这里推荐采用 yum 源方式进行安装。Centreon 的安装主要分为下面几个部分。

- 1) 系统基础依赖库的安装：例如 fping、cpp、gcc、gd、libjpeg-devel、rrdtool 等软件包。
- 2) rrdtool 的安装：安装 rrdtool 主要用于绘图，Centreon 利用 rrdtool 将收集到的数据

绘制成图形报表。

3) nagios、nagios-plugins 的安装：因为 nagios 是 Centreon 底层监控模块，所以 nagios 的安装是必不可少的。

4) ndoutils 的安装：ndoutils 是 nagios 与数据库进行连接的工具，它将 nagios 数据存入数据库，然后又将数据读取出来，它在 nagios 与 Centreon 之间接收和发送数据。

5) nrpe 的安装：nrpe 主要与远程的主机进行通信，进而收集远程主机的各种私有数据。

6) Centreon 的安装：这个才是要介绍的重点。Centreon 的官方网站是 <http://www.centreon.com/>，可以在这里下载最新的稳定版本，具体的安装过程后面会详细讲解。

这里我们采用的操作系统是 CentOS 5.5 x86_64 版本，下面开始安装 Centreon+nagios 监控系统。

8.4.1 安装支持 Centreon 的 yum 源

这里我们安装两个 yum 源，分别是 rpmforge 和 EPEL，可以从如下两个地址下载到：

□ <http://pkgs.repoforge.org/rpmforge-release/>

□ <http://fedoraproject.org/wiki/EPEL/zh-cn>

读者可根据自己的系统环境选择合适的版本下载，这里下载的是 rpmforge-release-0.5.3-1.el5.rf.x86_64.rpm 和 epel-release-5-4.noarch.rpm。下载完成，执行以下命令安装即可：

```
[root@centreon-server ~]# rpm -ivh rpmforge-release-0.5.3-1.el5.rf.x86_64.rpm
[root@centreon-server ~]# rpm -ivh epel-release-5-4.noarch.rpm
```

安装完成后，会在 /etc/yum.repos.d 下生成 yum 源的配置文件。

最后，还需要添加一个 yum 源，内容如下：

```
[root@centreon-server yum.repos.d]# more centreon.repo
[centreon]
name=Dag RPM Repository for Red Hat Enterprise Linux
baseurl=http://apt.sw.be/redhat/el$releasever/en/$basearch/dag
gpgcheck=1
gpgkey=http://dag.wieers.com/rpm/packages/RPM-GPG-KEY.dag.txt
enabled=1
```

将 centreon.repo 文件也放到 /etc/yum.repos.d 下即可，这个 yum 源中有我们需要的 nagios、ndoutils、nrpe 等软件包。

8.4.2 安装系统基础依赖库

(1) 安装 GD 库及 Apache

```
[root@centreon-server app]# yum install httpd gd fontconfig-devel libjpeg-devel
```



```
libpng-devel gd-devel perl-GD
```

(2) 安装 MySQL 库、PHP 及扩展

```
[root@centreon-server app]# yum install openssl-devel perl-DBD-MySQL mysql-server  
mysql-devel php php-mysql php-gd php-ldap php-xml php-mbstring
```

(3) 安装 Perl 及扩展模块

```
[root@centreon-server app]# yum install perl-Config-IniFiles perl-DBI perl-DBD-  
MySQL perl-Crypt-DES perl-Digest-SHA1
```

(4) 安装 SNMP 及依赖库

```
[root@centreon-server app]# yum install perl-Digest-HMAC net-snmp-utils perl-  
Socket6 perl-IO-Socket-INET6 net-snmp net-snmp-libs php-snmp
```

(5) 安装 rrdtool 绘图工具

```
[root@centreon-server app]# yum install rrdtool perl-rrdtool
```

(6) 安装其他所需库

```
[root@centreon-server app]# yum install dmidecode lm_sensors perl-Net-SNMP  
net-snmp-perl fping cpp gcc gcc-c++ libstdc++ glib2-devel
```

(7) 安装 PEAR

```
[root@centreon-server app]# yum install php-pear  
[root@centreon-server app]# pear channel-update pear.php.net
```

8.4.3 安装 nagios 及 nagios-plugins

有了上面三个 yum 源，安装 nagios 及插件会非常简单，操作如下：

```
[root@centreon-server app]# yum install nagios nagios-devel  
[root@centreon-server app]# yum install nagios-plugins
```

nagios 在 3.5 版本之后，将自带的所有插件都放在了另一个 RPM 包中，如果安装的是 nagios3.5 之上的版本，还需要安装 nagios-plugins-all 这个插件包，执行如下命令即可：

```
[root@centreon-server app]# yum install nagios-plugins-all
```

完成 nagios 安装后就会发现，插件安装路径是 /usr/lib64/nagios/plugins，而配置文件路径是 /etc/nagios，日志文件路径是 /var/log/nagios，这几个路径可能会在后面配置的时候用到。

8.4.4 安装 ndoutils

ndoutils 是必不可少的一部分，因为它是 nagios 与数据库进行连接的工具。有了前面的

yum 源，安装 ndoutils 变得非常简单，操作如下：

```
[root@centreon-server app]# yum install ndoutils
```

根据 ndoutils 版本的不同，可能还需要安装 ndoutils-mysql，执行如下命令即可：

```
[root@centreon-server app]# yum install ndoutils-mysql
```

完成 ndoutils 安装后可以发现，ndomod 模块安装路径是 /usr/libexec/ndomod-3x.o，或者 /usr/lib64/nagios/brokers/ndomod.so，而对应的维护脚本路径是 /etc/init.d/ndoutils 或 /etc/init.d/ndo2db。在本章介绍中，ndomod 模块安装路径是 /usr/libexec/ndomod-3x.o，而对应的维护脚本是 /etc/init.d/ndoutils，这些路径会在安装 Centreon 过程中用到。

8.4.5 安装 nrpe

如果需要获取远程主机的私有信息，就需要在监控主机上安装 check_nrpe 插件，同时在被监控主机上安装 nrpe 服务。安装 nrpe 的方法很简单，也可以通过 yum 源方式执行安装。

在监控主机上安装 check_nrpe 插件：

```
[root@centreon-server app]# yum install check_nrpe
```

在完成 check_nrpe 安装后，可以发现 check_nrpe 插件安装到了 /usr/lib64/nagios/plugins 目录下。

在被监控主机上安装 nrpe 服务：

```
[root@node1 app]# yum install nrpe
```

在完成 nrpe 安装后，默认的配置文件的目录是 /etc/nagios/nrpe.cfg，而维护 nrpe 的脚本路径是 /etc/init.d/nrpe。

8.4.6 安装 Centreon

首先到 Centreon 的官方网站下载最新稳定版本，这里我们下载的是 Centreon-2.4.1 版本，完成下载后开始安装：

```
[root@centreon-server app]# tar zxvf centreon-2.4.1.tar.gz
```

```
[root@centreon-server app]# cd centreon-2.4.1
```

```
[root@centreon-server centreon-2.4.1]# ./install.sh -i
```

下面就进入了交互安装界面，如图 8-2 所示。

安装 Centreon 第一步主要是检查必需的几个系统命令，然后会进入 GPL licence 确认，这里输入“y”即可进入下一步。

```

-----
Checking all needed binaries
-----
rm                                OK
cp                                OK
mv                                OK
/bin/chmod                        OK
/bin/chown                        OK
echo                              OK
more                              OK
mkdir                             OK
find                              OK
/bin/grep                         OK
/bin/cat                          OK
/bin/sed                          OK

Do you accept GPL license ?
[y/n], default to [n]:
> y

```

图 8-2 开始安装 Centreon

接下来这一步主要是确认需要安装的 Centreon 模块，主要有 Centreon Web Front、Centreon CentCore、Centreon Nagios Plugins、Centreon Snmp Traps process 四个模块可选，这里我们选择全部安装，如图 8-3 所示。

```

-----
Please choose what you want to install
-----
Do you want to install : Centreon Web Front
[y/n], default to [n]:
> y

Do you want to install : Centreon CentCore
[y/n], default to [n]:
> y

Do you want to install : Centreon Nagios Plugins
[y/n], default to [n]:
> y

Do you want to install : Centreon Snmp Traps process
[y/n], default to [n]:
> y

```

图 8-3 选择需要安装的 Centreon 模块

下面进入 Centreon Web Front 的安装过程，这里需要指定一系列安装路径，比如 Centreon 的安装目录、配置文件的安装路径、日志文件的目录等。安装过程会给出默认的安装路径，如果不需要更改路径，直接按回车键确认即可；如果需要改变默认的安装路径，输入自定义的路径，然后按回车键确认即可。读者可根据自己的环境，选择对应的安装目录，如图 8-4 和图 8-5 所示。

```

-----
Start CentWeb Installation
-----

Where is your Centreon directory?
default to [/usr/local/centreon]
>

Do you want me to create this directory ? [/usr/local/centreon]
[y/n], default to [n]:
> y
Path /usr/local/centreon                                OK

Where is your Centreon log directory
default to [/usr/local/centreon/log]
>

Do you want me to create this directory ? [/usr/local/centreon/log]
[y/n], default to [n]:
> y
Path /usr/local/centreon/log                            OK

Where is your Centreon etc directory
default to [/etc/centreon]
> /usr/local/centreon/etc

Do you want me to create this directory ? [/usr/local/centreon/etc]
[y/n], default to [n]:
> y
Path /usr/local/centreon/etc                            OK

```

图 8-4 Centreon Web 安装路径配置 1

```

Where is your Centreon binaries directory
default to [/usr/local/centreon/bin]
>

Do you want me to create this directory ? [/usr/local/centreon/bin]
[y/n], default to [n]:
> y
Path /usr/local/centreon/bin                            OK

Where is your Centreon data informations directory
default to [/usr/local/centreon/data]
>

Do you want me to create this directory ? [/usr/local/centreon/data]
[y/n], default to [n]:
> y
Path /usr/local/centreon/data                            OK

Where is your Centreon generation_files directory?
default to [/usr/local/centreon]
>
Path /usr/local/centreon                                OK

Where is your Centreon variable library directory?
default to [/var/lib/centreon]
>

Do you want me to create this directory ? [/var/lib/centreon]
[y/n], default to [n]:
> y
Path /var/lib/centreon                                  OK

```

图 8-5 Centreon Web 安装路径配置 2

需要注意的是，有些默认配置并不正确，此时，就需要手工指定某些库文件或模块的安装路径，例如，RRDs.pm 的安装路径、PEAR.php 的安装路径就需要通过手工来指定，如图 8-6 所示。


```

Where is your CentPlugins Traps binary
default to [/usr/local/centreon/bin]
>
Path /usr/local/centreon/bin OK

Where is the RRD perl module installed [RRDs.pm]
default to [/usr/lib/perl5/RRDs.pm]
> /usr/lib64/perl5/vendor_perl/5.8.8/x86_64-linux-thread-multi/RRDs.pm
Path /usr/lib64/perl5/vendor_perl/5.8.8/x86_64-linux-thread-multi
/usr/bin/rrdtool OK
/bin/mail OK
/usr/bin/php OK

Where is PEAR [PEAR.php]
default to [/usr/share/php/PEAR.php]
> /usr/share/pear/PEAR.php
Path /usr/share/pear OK
/usr/bin/perl OK
Finding Apache user : daemon
Finding Apache group : daemon

```

图 8-6 手工指定 Centreon 依赖文件安装路径

Centreon 支持多种管理引擎（也可称为监控引擎），例如 Centreon Engine、nagios 和 Icinga。可根据需要选择管理引擎，这里我们选择的是 nagios 管理引擎。同时 Centreon 也支持多种代理模块，例如 Centreon Broker 和 NDOUtils，这里采用的是 NDOUtils 代理模块。

这里需要特别注意的是，如果选择了 Centreon Engine 引擎，那么相应的“Monitoring engine user”就要选择“centreon-engine”；如果选择了 nagios 引擎，那么“Monitoring engine user”就要选择“nagios”。同理，如果选择了 Centreon Broker 代理模块，那么“Broker user”就要选择“centreon-broker”；如果选择了 NDOUtils 代理模块，那么“Broker user”就要选择“nagios”。

最后还需要给出管理引擎的日志目录及插件目录，整个过程如图 8-7 所示。

```

What is the Centreon group ? [centreon]
default to [centreon]
>

Do you want me to create this group ? [centreon]
[y/n], default to [n]:
> y

What is the Centreon user ? [centreon]
default to [centreon]
>

Do you want me to create this user ? [centreon]
[y/n], default to [n]:
> y

What is the Monitoring engine user ?
> nagios

What is the Broker user ? (optional)
> nagios

What is the Monitoring engine log directory ?
> /var/log/nagios

Where is your monitoring plugins (libexec) directory ?
default to [/usr/lib/nagios/plugins]
> /usr/lib64/nagios/plugins
Path /usr/lib64/nagios/plugins OK
Add group centreon to user daemon OK
Add group centreon to user nagios OK
Add group nagios to user daemon OK
Add group nagios to user centreon OK

```

图 8-7 配置 Centreon 的管理引擎和代理模块

在指定了对应的 Centreon 管理引擎和代理模块之后, 接下来, 就需要指定这些管理引擎和代理模块的维护脚本。nagios 管理引擎对应的维护脚本为 /etc/init.d/nagios, 由于 nagios 是 yum 源方式安装的, 因此对应的二进制文件为 /usr/bin/nagios。接着还需要指定管理引擎和代理模块的配置文件目录, 这里均为 /etc/nagios。最后指定代理模块的维护脚本路径为 /etc/init.d/ndoutils。在指定了所有的路径后, Centreon 安装程序会将这些管理脚本和路径统一写入 /etc/sudoers 文件中, 这是因为 Centreon 监控系统是在 Centreon 用户下运行的, 而这些维护脚本默认只有 root 用户才能执行, 因此要让 Centreon 用户统一来配置和维护, 就必须将这些维护脚本放入 /etc/sudoers 文件中, 实现无密码授权访问。

配置结果如图 8-8 所示。

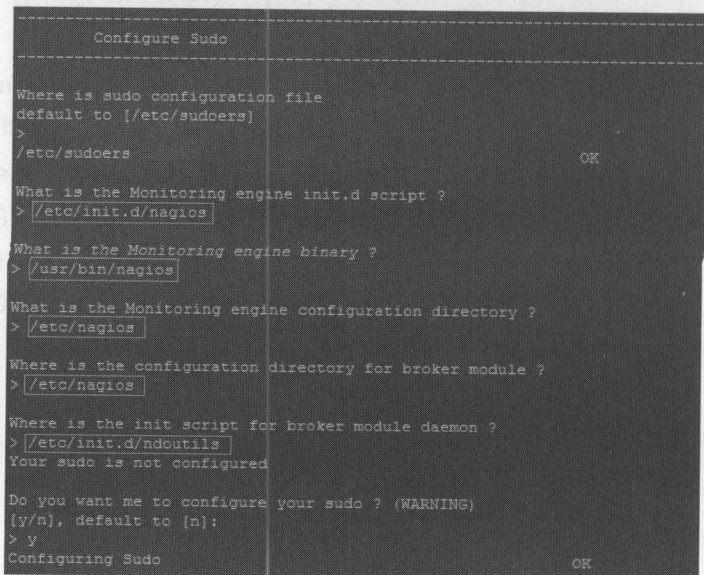


图 8-8 指定管理引擎和代理模块的维护脚本并添加到 sudoers 文件

下面的步骤是配置 Apache Server, 如图 8-9 和图 8-10 所示。由于 Centreon 是基于 Web 的一个应用, 默认使用的是 Apache Server, 因此安装程序会自动在 /etc/httpd/conf.d 下创建一个 centreon.conf 文件, 文件内容如下:

```
Alias /centreon /usr/local/centreon/www/
<Directory "/usr/local/centreon/www">
    Options Indexes
    AllowOverride AuthConfig Options
    Order allow,deny
    Allow from all
</Directory>
```

这样我们就可以通过 <http://ip/centreon> 的方式访问 Centreon 了。

接下来会提示是否重新加载 Apache 配置，最后会执行一系列动作，比如设置权限、复制配置文件、安装库文件等。每个动作执行后都会显示运行结果，如果都显示“OK”字样，表示所有动作执行正常。

```
-----
Configure Apache server
-----

Do you want to add Centreon Apache sub configuration file ?
[y/n], default to [n]:
> y
Create '/etc/httpd/conf.d/centreon.conf' OK
Configuring Apache OK

Do you want to reload your Apache ?
[y/n], default to [n]:
> y
Reloading Apache service OK
Preparing Centreon temporary files
Change right on /usr/local/centreon/log OK
Change right on /usr/local/centreon/etc OK
Change macros for insertBaseConf.sql OK
Change macros for sql update files OK
Change macros for php files OK
Change right on /etc/nagios OK
Add group nagios to user daemon OK
Add group nagios to user nagios OK
Add group centreon to user nagios OK
Copy CentWeb in system directory
Install CentWeb (web front of centreon) OK
Change right for install directory
Change right for install directory OK
Install libraries OK
Write right to Smarty Cache OK
Copying libinstall OK
Change macros for centreon.cron OK
Install Centreon cron.d file OK
Change macros for centAcl.php OK
Change macros for downtimeManager.php OK
Change macros for eventReportBuilder.pl OK
```

图 8-9 配置 Apache Server 并加载配置 1

```
Change macros for dashboardBuilder.pl OK
Install cron directory OK
Change right for eventReportBuilder.pl OK
Change right for dashboardBuilder.pl OK
Change macros for centreon.logrotate OK
Install Centreon logrotate.d file OK
Prepare export-mysql-indexes OK
Install export-mysql-indexes OK
Prepare import-mysql-indexes OK
Install import-mysql-indexes OK
Prepare indexes schema OK
Install indexes schema OK
```

图 8-10 配置 Apache Server 并加载配置 2

接下来是 PEAR 模块的检查和安装。PEAR 是运行 Centreon 必需的软件支持。安装进程会首先检查系统是否安装了这些必需的 PEAR 模块，以及版本是否正确，如果已经安装，会显示“OK”字样，如果没有安装，会显示“NOK”字样。对于缺少 PEAR 模块的情况，安装进程会提示是否在线进行安装或更新，这里选择“y”可进行在线更新，当然前提是服务器必须能连接互联网。紧接着安装进程就从互联网下载所需的软件包进行安装，直到安装完所有软件包，如图 8-11 所示。


```
-----
Pear Modules
-----
Check PEAR modules
PEAR                1.4.9          1.4.9          OK
DB                  1.7.6          NOK
DB_DataObject       1.8.4          NOK
DB_DataObject_FormBuilder 1.0.0RC4      NOK
MDB2                2.0.0          NOK
Date                1.4.6          NOK
HTML_Common         1.2.2          NOK
HTML_QuickForm      3.2.5          NOK
HTML_QuickForm_advmultiselect 1.1.0          NOK
HTML_Table          1.6.1          NOK
Archive_Tar         1.1            1.3.1          OK
Auth_SASL           1.0.1          NOK
Console_Getopt      1.2            1.2            OK
Net_SMTP            1.2.8          NOK
Net_Socket          1.0.1          NOK
Net_Traceroute      0.21           NOK
Net_Ping            2.4.1          NOK
Validate            0.6.2          NOK
XML_RPC             1.4.5          1.5.5          OK
SOAP                0.10.1         NOK
Log                 1.9.11         NOK
Archive_Zip         0.1.2          NOK

Do you want me to install/upgrade your PEAR modules
[y/n], default to [y]:
> y
Upgrading PEAR modules
Installing PEAR modules
```

图 8-11 在线安装 PEAR 模块

下面这个步骤是创建两个配置文件，为安装 Centreon 做准备，其实也就是创建安装 Centreon 的文件模板，如图 8-12 所示。

```
-----
Centreon Post Install
-----
Create /usr/local/centreon/www/install/install.conf.php OK
Create /usr/local/centreon/etc/instCentWeb.conf OK
-----
```

图 8-12 创建安装 Centreon 的文件模板

接下来正式进入 Centreon 的安装过程。Centreon 的安装分为安装 CentStorage、安装 CentCore、安装 CentPlugins、安装 CentPlugins Traps 四个部分。

首先是 CentStorage 的安装，如图 8-13 和图 8-14 所示，这个过程会询问 Centreon 的运行目录、CentStorage 的二进制文件目录、CentStorage 的 rrd 数据存放目录等，分别指定路径后，安装进程将会创建相应目录。

图 8-14 演示的主要是安装 CentStorage 管理维护脚本及设定运行级等。

接下来是安装 CentCore。与 CentStorage 的安装类似，安装进程会创建相应的 CentCore 运行目录、维护脚本等，如图 8-15 所示。

然后是安装 CentPlugins，这里指定 CentPlugins 的安装目录是 /var/lib/centreon/centplugins，如图 8-16 所示。


```

-----
Start CentStorage Installation
-----

Where is your Centreon Run Dir directory?
default to [/var/run/centreon]
>

Do you want me to create this directory ? [/var/run/centreon]
[y/n], default to [n]:
> y
Path /var/run/centreon OK

Where is your CentStorage binary directory
default to [/usr/local/centreon/bin]
>
Path /usr/local/centreon/bin OK

Where is your CentStorage RRD directory
default to [/var/lib/centreon]
>
Path /var/lib/centreon OK

Preparing Centreon temporary files
/tmp/centreon-setup exists, it will be moved...
install www/install/createTablesCentstorage.sql OK
Creating Centreon Directory '/var/lib/centreon/status' OK
Creating Centreon Directory '/var/lib/centreon/metrics' OK
Change macros for centstorage binary OK
Install CentStorage binary OK
Install library for centstorage OK
Change right : /var/run/centreon OK
Change macros for centstorage init script OK

```

图 8-13 安装 CentStorage 并创建指定目录

```

Do you want me to install CentStorage init script ?
[y/n], default to [n]:
> y
CentStorage init script installed OK

Do you want me to install CentStorage run level ?
[y/n], default to [n]:
> y
Change macros for logAnalyser OK
Install logAnalyser OK
Change macros for logAnalyser-cbroker OK
Install logAnalyser-cbroker OK
Change macros for nagiosPerfTrace OK
Install nagiosPerfTrace OK
Change macros for purgeLogs OK
Install purgeLogs OK
Change macros for purgeCentstorage OK
Install purgeCentstorage OK
Change macros for centreonPurge.sh OK
Install centreonPurge.sh OK
Change macros for centstorage.cron OK
Install CentStorage cron OK
Change macros for centstorage.logrotate OK
Install Centreon Storage logrotate.d file OK
Create /usr/local/centreon/etc/instCentStorage.conf OK

```

图 8-14 安装 CentStorage 并创建维护脚本

```

-----
Start CentCore Installation
-----

Where is your CentCore binary directory
default to [/usr/local/centreon/bin]
>
Path /usr/local/centreon/bin                                OK
/usr/bin/ssh                                                OK
/usr/bin/scp                                                OK
Preparing Centreon temporary files
/tmp/centreon-setup exists, it will be moved...
Change CentCore Macro                                      OK
Copy CentCore in binary directory                          OK
Change right : /var/run/centreon                            OK
Change right : /var/lib/centreon                            OK
Change macros for centcore.logrotate                       OK
Install Centreon Core logrotate.d file                     OK
Replace CentCore init script Macro                         OK

Do you want me to install CentCore init script ?
[y/n], default to [n]:
> y
CentCore init script installed                             OK

Do you want me to install CentCore run level ?
[y/n], default to [n]:
> y
Create /usr/local/centreon/etc/instCentCore.conf           OK

```

图 8-15 安装 CentCore

```

-----
Start CentPlugins Installation
-----

Where is your CentPlugins lib directory
default to [/var/lib/centreon/centplugins]
>

Do you want me to create this directory ? [/var/lib/centreon/centplugins]
[y/n], default to [n]:
> y
Path /var/lib/centreon/centplugins                          OK
Preparing Centreon temporary files
/tmp/centreon-setup exists, it will be moved...
Change macros for CentPlugins                              OK
Installing the plugins                                      OK
Change right on centreon.conf                              OK
CentPlugins is installed

```

图 8-16 安装 CentPlugins

最后是 CentPlugins Traps 的安装。这个过程会要求指定 snmp 配置目录, SNMPTT 的存放目录等, 然后安装进程会将配置文件分别放到指定的目录下, 如图 8-17 所示。

至此, Centreon 的安装完成, 下面就可以进入 Centreon Web 的安装配置。不过在进行 Centreon Web 的安装配置之前, 需要通过以下命令启动相关的几个服务:

```

[root@centreon-server app]# /etc/init.d/httpd start
[root@centreon-server app]# /etc/init.d/mysqld start
[root@centreon-server app]# /etc/init.d/ndoutils start

```

```

-----
Start CentPlugins Traps Installation
-----

Where is your SNMP configuration directory
default to [/etc/snmp]
>
/etc/snmp                                     OK

Where is your SNMPTT binaries directory
default to [/usr/local/centreon/bin/]
>
/usr/local/centreon/bin/                     OK
Finding Apache user :                         daemon
Preparing Centreon temporary files
/tmp/centreon-setup exists, it will be moved...
Change macros for CentPluginsTraps           OK
Change macros for init scripts               OK
Installing the plugins Trap binaries         OK
Change macros for snmptrapd.conf             OK
Change macros for snmptt.ini                 OK
SNMPTT init script installed                 OK
Install : snmptrapd.conf                     OK
Install : snmp.conf                          OK
Install : snmptt.ini                         OK
Install : snmptt                             OK
Install : snmptthandler                      OK
Install : snmpttconvertmib                   OK
Create /usr/local/centreon/etc/instCentPlugins.conf OK

```

图 8-17 安装 CentPlugins Traps

8.4.7 安装配置 Centreon Web

Centreon 提供了非常友好的 Web 安装向导界面，通过这个界面可以一步一步完成 Centreon 的最后安装过程。

首先在浏览器中输入访问 Centreon Web 的地址，例如：<http://centreon-server/centreon>。如果上面的安装过程都正确，就会出现如图 8-18 所示的欢迎界面。

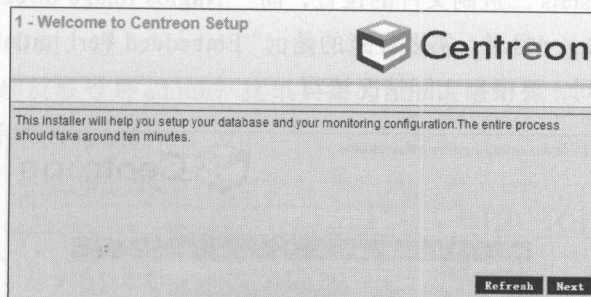


图 8-18 Centreon Web 欢迎页面

单击“Next”按钮，进入下一步，如图 8-19 所示，这一步主要是验证一些必需的依赖模块，例如 mysql.so、gd.so、ldap.so 等是否正常加载。

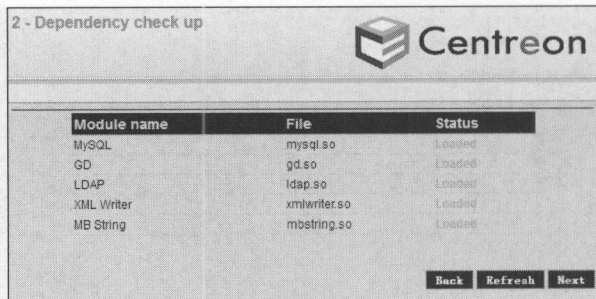


图 8-19 检测必需的依赖模块

单击“Next”按钮，进入下一步，如图 8-20 所示。这一步主要是选择一个用于 Centreon 的管理引擎，这里选择 nagios。

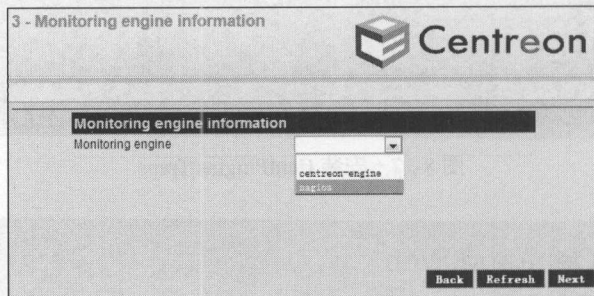


图 8-20 选择管理引擎

单击“Next”按钮，进入下一步，如图 8-21 所示。这一步主要是指定管理引擎 nagios 对应的一些配置信息，其中“Nagios directory”是指定 nagios 的主目录，“Nagiostats binary”是指定 nagiosstats 二进制文件的位置，而“Nagios image directory”对应的目录是 nagios 主目录下的 images 目录。需要注意的是，“Embedded Perl initialisation file”对应的路径可能因环境而不同，要根据实际情况填写。

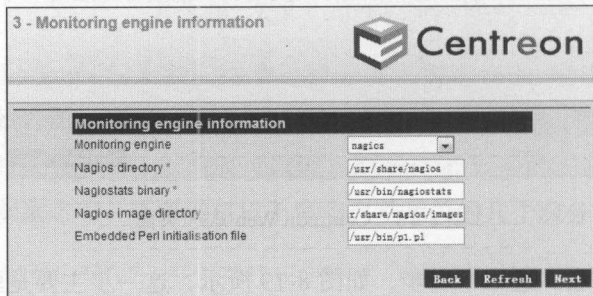


图 8-21 指定管理引擎对应的信息

单击“Next”按钮，进入下一步，如图 8-22 所示。这一步是选择 Centreon 使用的代理模块，由于前面已经配置了 NDOUtils 作为代理模块，因此这里选择“nodutils”。

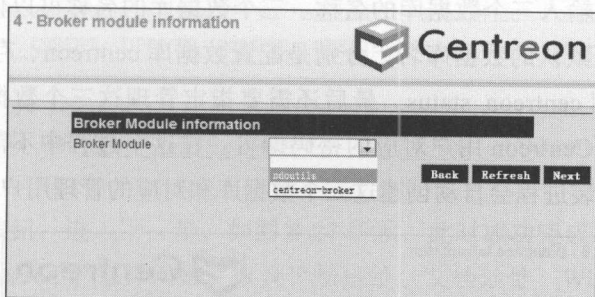


图 8-22 选择代理模块

单击“Next”按钮，进入下一步，如图 8-23 所示。这一步是指定代理模块 NDOUtils 的配置信息，其实就是指定 ndomod.o 的路径，这个路径会因为安装 NDOUtils 方式的不同而有差异，读者可根据自己的实际情况填写。

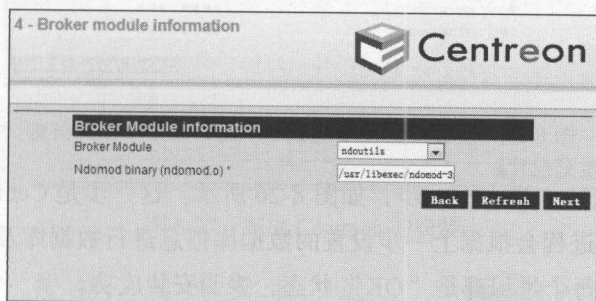


图 8-23 指定代理模块配置信息

单击“Next”按钮，进入下一步，如图 8-24 所示。这一步是添加管理员账号信息，Centreon 默认的管理员账号是 admin，这里只需为 admin 账号创建一个密码，然后输入账号的其他相关信息和邮件地址即可。

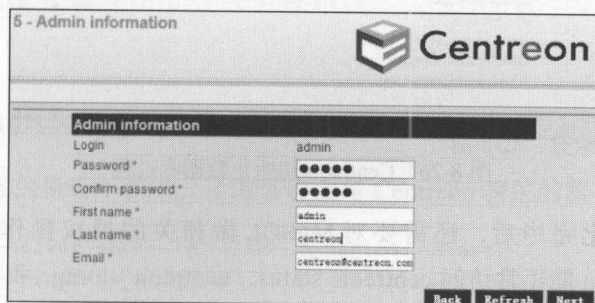


图 8-24 设置管理员账号密码

单击“Next”按钮，进入下一步，如图8-25所示。这一步是设置 Centreon 连接 MySQL 数据库的相关信息，在这里输入 MySQL 数据库的 IP 地址、端口号、root 用户密码等信息，然后还要输入三个数据库的名称。三个数据库的名称可以使用默认的，也可以自行指定，这里采用默认的数据库名，分别是配置数据库 centreon、存储数据库 centreon_storage、Utils 数据库 centreon_status。最后还需要指定管理这三个数据库的用户，默认是 Centreon，然后添加 Centreon 用户对应的密码即可。在这个过程中不需要手工创建这三个数据库，Centreon 安装进程会自动创建这三个数据库和对应的管理用户。

6 - Database information

Database information

Database Host Address (default: localhost)

Database Port (default: 3306)

Root password

Configuration database name *

Storage database name *

Utils database name *

Database user name *

Database user password *

Confirm user password *

Back Refresh Next

图 8-25 设置 Centreon 连接 MySQL 数据库相关信息

单击“Next”按钮，进入下一步，如图8-26所示。这一步是 Centreon 初始化 MySQL 数据库的过程，安装进程会根据上一步设置的数据库信息进行数据库及数据库用户的创建。如果初始化数据库的每个过程都是“OK”状态，表明安装成功。

7 - Installation

Currently installing database... please do not interrupt this process.

Step	Status
Configuration database	OK
Storage database	OK
Utils database	OK
Creating database user	OK
Setting up basic configuration	OK
Setting up configuration file	OK

Next

图 8-26 Centreon 初始化数据库过程

在数据库初始化完毕后，还需要对 MySQL 做相关的授权操作，以使上面指定的 MySQL 用户 Centreon 能正常访问 centreon_status、centreon_storage 和 centreon 这三个数据库。授权方法很简单，通过命令行登录 MySQL 数据库中，执行如下操作：

```
SQL>GRANT USAGE ON `centreon_status`.* TO 'centreon'@'localhost' IDENTIFIED BY 'centreon';
SQL>GRANT SELECT, INSERT, UPDATE, DELETE, CREATE ON `centreon_status`.* TO 'centreon'@'localhost';
SQL>GRANT USAGE ON `centreon_storage`.* TO 'centreon'@'localhost' IDENTIFIED BY 'centreon';
SQL>GRANT SELECT, INSERT, UPDATE, DELETE, CREATE ON `centreon_storage`.* TO 'centreon'@'localhost';
SQL>GRANT USAGE ON `centreon`.* TO 'centreon'@'localhost' IDENTIFIED BY 'centreon';
SQL>GRANT SELECT, INSERT, UPDATE, DELETE, CREATE ON `centreon`.* TO 'centreon'@'localhost';
```

单击“Next”按钮，进入下一步，如图 8-27 所示，此时成功完成 Centreon 的安装。这个界面还展示了 Centreon 的官方网站，以及论坛地址、文档地址、Wiki 地址等信息，读者可以在这些站点获得更多关于 Centreon 的信息。

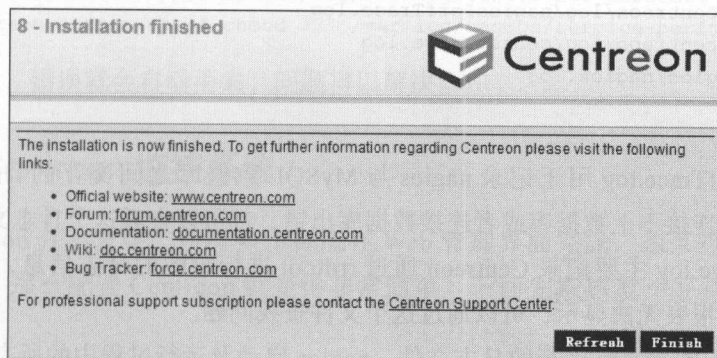


图 8-27 完成 Centreon 的安装

单击“Finish”按钮，完成安装，进入 Centreon 登录界面，如图 8-28 所示。登录 Centreon 管理界面的地址一般是 `http://IP/centreon`，输入用户名和密码即可登录 Centreon 监控平台。

至此，关于 Centreon 的安装部分介绍完毕。下面介绍如何快速配置和使用分布式监控系统 Centreon。



图 8-28 登录 Centreon 监控平台

8.4.8 启动 Centreon 相关服务

在完成 Centreon 的所有工作后，还需要启动与 Centreon 相关的服务，执行如下操作：

```
[root@centreon-server app]# /etc/init.d/nagios start
[root@centreon-server app]# /etc/init.d/centstorage start
[root@centreon-server app]# /etc/init.d/centcore start
```

如图 8-29 所示，这是对 Centreon-

在服务启动后,可以通过查看 nagios 启动日志 /var/log/nagios/nagios.log 及 Centreon 日志目录 /usr/local/centreon/log 来查看启动进程的运行状态。

8.4.9 安装问题总结

Centreon 的安装相对较复杂,牵扯的内容较多,在安装过程中出现问题在所难免,这里就简单总结下安装过程中可能出现的问题和解决方法。

在安装 Centreon 过程中,常见问题主要集中在 nagios 和 MySQL 上面,因此需要重点关注这两个方面。解决问题的主要方法是查看运行日志,因为所有错误都会在日志中显示,需要重点关注的日志有如下三个:

```
/usr/local/centreon/log/nagiosPerfTrace.log
/usr/local/centreon/log/centstorage.log
/var/log/nagios/nagios.log
```

其中:

- ❑ nagiosPerfTrace.log 用于记录 nagios 与 MySQL 数据库之间交互的日志信息,如果 Centreon 连接不上数据库或者连接数据库出错,都可以查看这个日志文件。
- ❑ centstorage.log 主要记录 Centreon 通过 rrdtool 进行绘图的日志信息,如果 Centreon 中的图形报表无法显示,可以通过这个文件查找问题。
- ❑ nagios.log 是 nagios 运行的日志文件,nagios 启动及运行过程中的所有信息都会在此文件中记录,如果启动或运行 nagios 出错,都可以通过查看这个文件来解决问题。

下面介绍 Centreon 安装过程中几个最常见的错误及相应的解决方案。

(1) nagios 能启动,但是不能正常工作

这个问题常见的现象是:

```
Error: Could not create external command file '/var/log/nagios/rw/nagios.cmd' as
named pipe: (2) -> No such file or directory
```

从错误提示中可以看出,是 nagios 不能自动创建 nagios.cmd 文件导致的,进入 /var/log/nagios 目录下发现确实不存在 rw 目录。

nagios.cmd 这个文件是用来检查外部命令请求的,也是用户操作 CGI 命令写入的地方,因此这个文件必须对相关的服务具有可写权限,一般是针对启动 Web 服务的用户,例如 daemon 用户,并且需要注意的是,这个文件所在的目录对 daemon 用户必须可写,否则会报上面这个错误。

在了解产生这个问题的原因后,能够采用的解决方法很多,最简单的方法是在 /var/log/nagios 目录下创建 rw 目录,然后授权即可,操作如下:


```
[root@centreon-server app]# mkdir /var/log/nagios/rw
[root@centreon-server app]# chmod 777 /var/log/nagios/rw
```

这样，在重启 nagios 服务后，就会自动在 /var/log/nagios/rw 下创建 nagios.cmd 文件。

(2) Centreon 下服务状态无法生成曲线图

这个问题常见的现象是：

```
2013-10-16 14:08:57 - Cannot write the /var/log/nagios/service-perfdata_read file : 权限不够
2013-10-16 14:08:57 - Error When writing data in tmp read file : 对设备不适当的 ioctl 操作
```

这个错误是没有权限写文件，进入 /var/log/nagios 目录下发现根本无此文件，需要手动创建此文件，然后进行授权，操作如下：

```
[root@centreon-server app]# touch /var/log/nagios/service-perfdata_read
[root@centreon-server app]# chmod 777 /var/log/nagios/service-perfdata_read
```

这样操作后，错误就会自动消失，问题得以解决。

8.5 配置 Centreon 监控系统

配置 Centreon 并不复杂，所有操作都能在 Web 管理界面完成，如果对 nagios 的配置过程比较了解，那么配置 Centreon 就变得非常简单。下面先来熟悉一下 nagios 中配置文件之间的关系。

在 nagios 的配置过程中涉及几个定义：主机、主机组、主机模板，服务、服务组、服务模板，联系人、联系人组、监控时间和监控命令等，从这些定义可以看出，nagios 的各个配置文件之间是互为关联，彼此引用的。成功配置一台 nagios 监控系统，必须弄清楚每个配置文件之间依赖与被依赖的关系，其中，最重要的有四点：第一要定义监控哪些主机、主机组、服务和组；第二要定义这个监控要通过什么命令实现；第三要定义监控的时间段；第四要定义主机或服务出现问题时通知的联系人和联系人组。

Centreon 的配置逻辑和过程与 nagios 完全相同，因此，清楚了 nagios 的配置重点和各个配置文件之间的依赖关系，Centreon 的配置将变得比 nagios 更加简单。

8.5.1 添加主机和主机组

默认情况下，Centreon 在安装完成后，会有一些初始的主机或服务的监控项，为了方便介绍和演示，这里删除所有服务的默认监控：选择“Configuration → Services → Services by host”，删除所有默认监控的服务即可。

1. Centreon-Server 主机的定义

选择“Configuration → Hosts → Centreon-Server”，如图 8-29 所示，这是对 Centreon-

Server 主机的定义，其中，“HostName”项就是要监控节点的主机名。“IP Address/DNS”项可以是添加的 IP 地址，也可以是添加主机对应的主机名，只要 DNS 能解析到对应的 IP 就行。“Host Templates”项是为 Centreon-Server 添加的一个主机模板，而“generic-host”模板是默认就存在的，这里可以直接引用。关于这个主机模板，后面会有更详细的介绍。

对于一台主机的定义，基本上需要关注的就是以上几个选项，其他选项会陆续讲解。

The screenshot shows the 'Configuration > Hosts > Centreon-Server' interface. It has four tabs: 'Host Configuration', 'Relations', 'Data Processing', and 'Host Extended Infos'. The 'Host Configuration' tab is selected. Below the tabs is a 'Modify a Host' section. Under this, there's a 'General Information' section with the following fields:

- Host Name *: Centreon-Server
- Alias: Monitoring Server
- IP Address / DNS *: 127.0.0.1 (with a 'Resolve' button)
- SNMP Community & Version: (empty field with a dropdown arrow)
- Monitored from: Central (dropdown menu)
- Host Templates: A host can have multiple templates, their orders have a significant importance. Here is a self explanatory image. Below this text is 'Add a template +' and a dropdown menu showing 'generic-host' with a refresh icon.
- Create Services linked to the Template too: Yes (radio button) and No (radio button, which is selected).

图 8-29 Centreon-Server 主机的定义

2. 创建一个 check_ping 命令

监控命令（Commands）是 Centreon 分布式监控系统运行的基础，无论主机还是服务，都是通过监控命令完成状态检查和报警的。经常使用的监控命令分为两种，分别是检测（Check）命令和通知告警（Notification）命令。在 Centreon 系统中，默认自带了很多常用的监控命令，选择“Configuration → Commands → Checks”，即可看到 Centreon 自带的监控检测命令，例如 check_dhcp、check_ftp、check_centreon_ping、check_http、check_tcp 等。选择“Configuration → Commands → Notifications”，即可看到自带的通知告警命令，例如常用的 host-notify-by-email、service-notify-by-email 等。

选择“Configuration → Commands”，单击“add”创建一个命令，如图 8-30 所示，其中，“Command Name”为 check_ping；“Command Type”为“Check”；“Command Line”是命令的具体执行方式，命令中“\$USER1\$”是一个变量，其实就是 nagios 插件的存放路径，“\$HOSTADDRESS\$”是个主机宏，用于取主机定义里的 IP 地址或者主机名，这些内容的含义与在 nagios 下表示的含义完全相同，这里不再过多介绍。

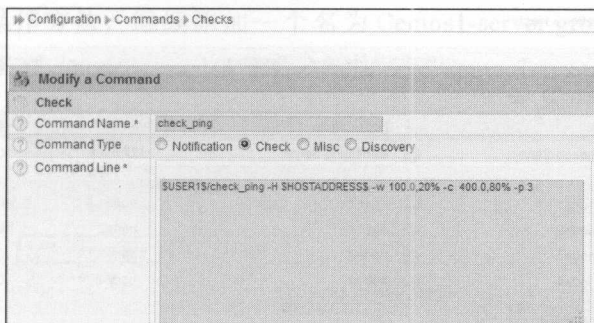


图 8-30 添加一个 check_ping 命令

3. 添加主机监控

选择“Configuration → Hosts”，单击“Add”添加一台主机，如图 8-31 所示。添加的方法与前面针对 Centreon-Server 主机的定义完全相同，首先添加一台 node1 主机，此主机引用的模板仍然是“generic-host”，最后，在“Host Check Properties”项中添加一个用于检测主机状态的命令“check_ping”，这个命令在上面的步骤中已经定义好了，这里直接引用即可。

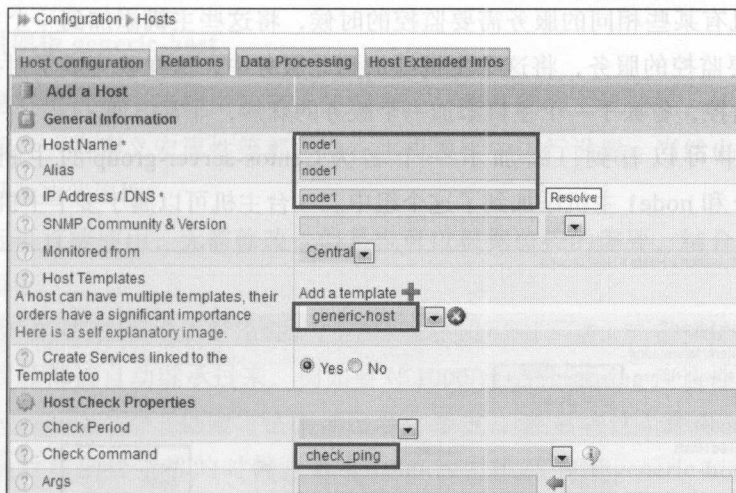


图 8-31 添加 node1 主机监控

采用与添加 node1 主机完全相同的方法，下面依次添加 node2、node3、node4 三台主机，如图 8-32 所示，这里一共添加了 4 台主机。在这个界面上，有很多操作属性，可以用于对主机进行复制、删除、修改、启用和禁用等，由此可见，通过 Centreon 管理 nagios 非常方便和简单。

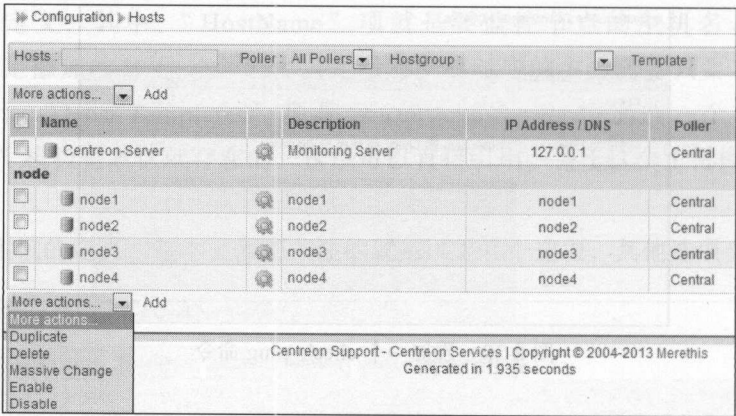


图 8-32 主机监控列表

4. 添加主机组监控

选择“Configuration → Hosts → Host Groups”，单击“Add”添加一个主机组，如图 8-33 所示，依次输入“Host Group Name”和“Alias”的值，然后在“Linked Hosts”中选择此主机组需要加入的主机即可，添加完毕，单击“Save”即可完成主机组的添加。

当一批主机有某些相同的服务需要监控的时候，将这些主机添加到一个主机组中，然后创建一个需要监控的服务，将这个主机组加到此服务中，这样就完成了对批量主机的某些相同服务的监控，省去了一个主机添加一个服务的麻烦，非常方便。

从图 8-33 中可以看到，添加了一个名为 Centos-server-group 的主机组，然后将 Centreon-Server 和 node1 主机添加到了这个组中。一台主机可以属于多个主机组。

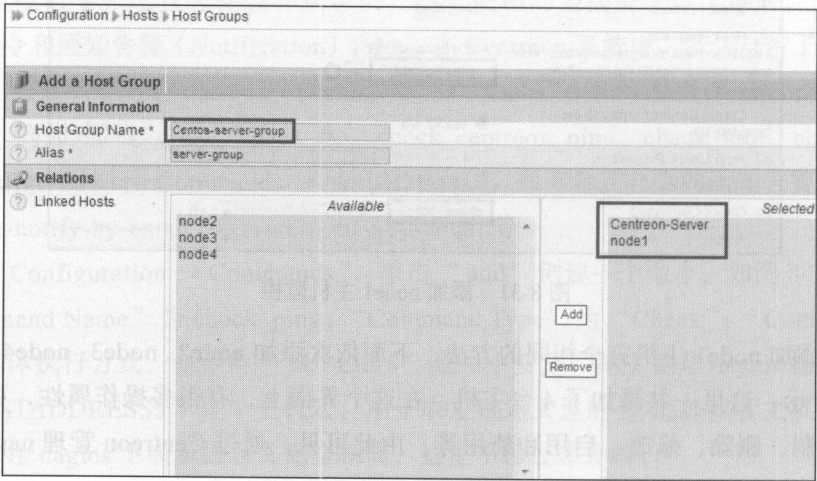


图 8-33 添加 Centos-server-group 主机组

根据图 8-33 的操作方法, 继续添加一个名为 Centos1-server-group 的主机组, 如图 8-34 所示, 这里将 node2、node3、node4 三台主机加入到 Centos1-server-group 主机组中。

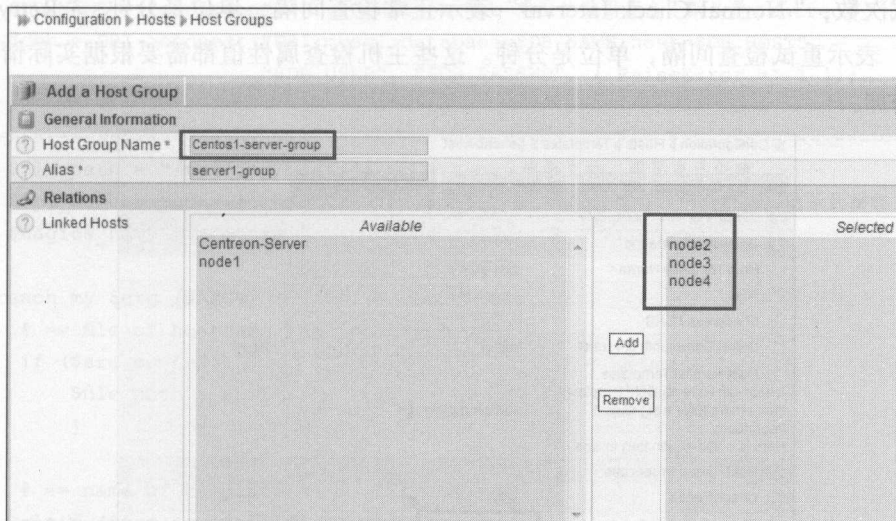


图 8-34 添加 Centos1-server-group 主机组

5. 配置主机模板 generic-host

主机模板是对主机各个监控属性定义的综合, 一些基础的主机监控, 比如主机检查属性、报警通知属性、自定义宏属性等都可以在主机模板中进行设置, 当然也可以在定义主机监控的时候设置这些属性。

默认此模板可直接使用, 无需修改, 但是也可以根据监控的需要, 结合自身环境修改默认主机监控模板。

主机模板的一个最大特点是继承性, 如果一台主机引用了这个模板, 那么此主机模板下的所有监控属性都被自动继承过来, 例如要对 1000 台主机做 ping 连通性检查, 首先可以创建一个 check_ping 命令 (创建方法上面已经介绍过), 然后将这个命令引用到 generic-host 模板中, 最后在创建主机的时候, 所有 1000 台主机都引用 generic-host 这个主机模板即可, 如图 8-35 所示。引用主机模板的好处是, 如果监控属性发生了变化, 只需修改 generic-host 配置即可, 而无须一台主机一台主机修改, 方便快捷。

有时候可能会发生一个监控属性既在 generic-host 模板中设置了, 也在主机定义中设置了, 此时就有一个优先级的问題, 在这种情况下, 监控属性的生效值以主机中的设置为准。例如, 已经在 generic-host 中设置了 Check Period 为 “24x7”, 而在 node1 主机的定义中也引用了 generic-host 模板, 同时将 Check Period 设置改为 “workhours”, 那么 node1 主机监

控周期最终生效的设置是“workhours”。

在图 8-35 中, 可以看到一些主机检查属性值, 例如, “Max Check Attempts”表示最大检查尝试次数; “Normal Check Interval”表示正常检查间隔, 单位是分钟; “Retry Check Interval”表示重试检查间隔, 单位是分钟。这些主机检查属性值都需要根据实际情况进行修改或添加。

The screenshot shows the 'generic-host' template configuration page. The 'Host Check Properties' section is expanded, displaying the following settings:

- Check Period: 24x7
- Check Command: check_ping
- Args: (empty)
- Max Check Attempts: 5
- Normal Check Interval: * 60 seconds
- Retry Check Interval: * 60 seconds
- Active Checks Enabled: Yes (selected)
- Passive Checks Enabled: No (selected)

图 8-35 修改 generic-host 主机模板

8.5.2 批量添加主机

添加一台主机非常简单, 但是如果有成千上万台主机需要添加, 一台台添加显然是不行的, 此时就需要通过批量添加主机的方法实现。这其实是借助 Centreon 的模板功能完成的, 其基本原理是: 先把批量主机的共同属性添加到主机模板中, 然后在批量添加主机时, 引用这个主机模板即可。这样在添加每台主机时不同的属性只有 IP 地址和主机名, 只要把这两个值写入数据库即可完成主机的添加。

下面是一个写好的批量添加主机的 Perl 脚本:

```
#!/usr/bin/perl
use strict;
use warnings;
use DBI;
use DBD::mysql;

# -----
```

```

my $DB_HOST = "127.0.0.1"; # 监控服务器的 IP 地址, 建议修改为 127.0.0.1
my $DB_USER = "centreon"; # Centreon Web 安装时设置的数据库访问用户, 默认为 centreon
my $DB_PASSWD = "centreon"; # Centreon Web 安装时设置的数据库密码, 这里为 centreon
my $DB_NAME = "centreon"; # Centreon Web 对应的数据库名, 默认是 centreon
my $dbh = DBI->connect("DBI:mysql:database=$DB_NAME;host=$DB_HOST",
    "$DB_USER", "$DB_PASSWD", { RaiseError => 1 });

```

```

# -----

```

```

my $file_path = "/var/tmp/hosts"; # hosts 模板文件, 需要自己创建
my $tpl_name = "generic-host"; # 主机模板, 填写批量添加的主机需要继承的模板
my $nagios_name = "Central"; # poller, 默认为 Central

```

```

foreach my $arg (@ARGV) {
    # == file of hostname and ipaddress ==
    if ($arg eq '-f') {
        $file_path = shift;
    }

    # == name of template ==
    elsif ($arg eq '-t') {
        $tpl_name = shift;
    }

    # == name of nagios name ==
    elsif ($arg eq '-n') {
        $nagios_name = shift;
    }

    else {
        &print_help();
        exit 1;
    }
}

```

```

# -----

```

```

open (HOST, "$file_path") || die "Cannot open $file_path for read";

```

```

my $sql;
my $sth;
my $line;
my ($host, $ipaddr);
my ($host_id, $tpl_id, $nagios_id) = (0, 0, 0);

```

```

while (defined($line = <HOST>)) {

```

```

    # == skip blank lines =====

```

```

next if ($line =~ /^\\s*$/);

# == skip if # =====
next if ($line =~ /^\\s*#/);

# == get host and ipaddress =====
($ipaddr, $host) = split(/\\s+/, $line);
next if ($ipaddr eq '' || $host eq '');

# == insert the host to table host ====
$sql = "insert host set
    host_template_model_htm_id='2',host_name='$host',host_alias='$host',host_
address='$ipaddr',host_active_checks_enabled='2',host_passive_checks_
enabled='2',host_checks_enabled='1',host_event_handler_enabled='2',host_
flap_detection_enabled='2',host_process_perf_data='2',host_retain_status_
information='2',host_retain_nonstatus_information='2',host_notifications_
enabled='2',host_register='1',host_activate='1'";

    $sth = $dbh->do($sql);
    sleep(1);

    # == get host_id =====
    $sql = "select host_id from host where host_name='$host'";
    $sth = $dbh->prepare($sql);
    $sth->execute();

    while (my $ref = $sth->fetchrow_hashref()) {
        $host_id = $ref->{'host_id'};
        print "host_id is $host_id\\n";
    }

    next if ($host_id == 0);

    # == insert extended_host_information ==
    $sql = "insert extended_host_information set host_host_id='$host_id'";
    $sth = $dbh->do($sql);

    # == insert host_template_relation =====
    $sql = "select host_id from host where host_name='$tpl_name'";
    $sth = $dbh->prepare($sql);
    $sth->execute();

    while (my $ref = $sth->fetchrow_hashref()) {
        $tpl_id = $ref->{'host_id'};
        print "template id is $tpl_id\\n";
    }

```



```

next if ($tpl_id == 0);

$sql = "insert host_template_relation set host_host_id='$host_id',host_tpl_
id='$tpl_id','order`='1'";
$stmt = $dbh->prepare($sql);
$stmt->execute();

# == insert ns_host_relation =====
$sql = "select id from nagios_server where name='$nagios_name'";

$stmt = $dbh->prepare($sql);
$stmt->execute();

while (my $ref = $sth->fetchrow_hashref()) {
    $nagios_id = $ref->{'id'};
    print "Poller id is $nagios_id\n";
}

next if ($nagios_id == 0);

$sql = "insert ns_host_relation set host_host_id='$host_id',nagios_server_
id='$nagios_id'";
$stmt = $dbh->prepare($sql);
$stmt->execute();

# == insert complete ==
print "insert $host to centreon successful\n";
}

close($HOST);
$dbh->disconnect();
exit 0;

# -----
sub print_help {
    print "Usage ./batch_add_host.pl [-f path of host file] [-n nagios name] [-t
template name]\n";
    print "\n";
}

```

要使用这个 Perl 脚本，需要具备如下条件：

- 1) 一个已经配置好的主机模板文件，在上面脚本中为 generic-host。
- 2) 一个 IP 和主机名对应的 hosts 文件，并且需要放到 /var/tmp 目录下。

hosts 文件的内容格式为“IP 地址.主机名”，每行一个，将多个主机依次添加到这个 hosts 文件中即可。将上面脚本命名为 batch_add_host.pl，放到监控服务器任意路径下，授予其可执行权限，然后执行这个脚本即可完成主机的批量添加。

对于批量添加的主机，“Host Name”选项对应 hosts 文件中的主机名，“IP Address / DNS”选项对应 hosts 文件的 IP 地址。

8.5.3 监控引擎管理

在完成主机和主机组添加后，这些主机信息并不会马上生效，还需要将这些信息生成 nagios 配置文件进行保存，然后重新启动监控引擎，这就是 Centreon 的监控引擎管理功能。在任何配置添加或修改完成后，都需要重启监控引擎才能使这些配置生效。

选择“Configuration → Monitoring Engines → Generate”，如图 8-36 所示。

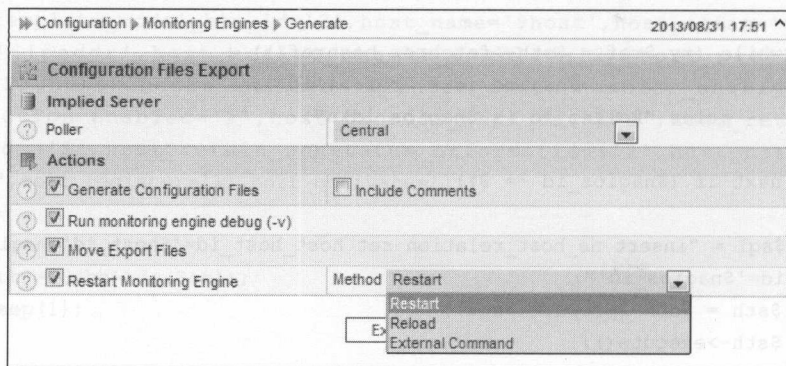


图 8-36 生成配置并重启监控引擎

图 8-36 主要用于导出创建好的配置，默认选择“Generate Configuration Files”和“Run monitoring engine debug (-v)”两个动作，当然也建议选择后面两项。在“Restart Monitoring Engine”一项中，可选的动作有三个，分别是 Restart、Reload 和 External Command，其中 Restart 用于重新启动监控引擎服务，例如新增加了一台主机，就可以使用 Restart 这个动作；而 Reload 是重新加载配置，例如修改了某台主机的配置参数，就可以使用 Reload 重新加载；而 External Command 用于第三方扩展，主要用于特殊需求，一般用不到。在通常情况下，Restart 动作执行的时间较长，特别是当 Centreon 监控的主机或服务较多时，执行 Restart 操作会启动得比较慢，而 Reload 操作仅仅是将新的配置加载生效，执行速度相对较快，因此，如何选择这两个动作参数，要结合实际情况而定。

在监控引擎管理中，还可以对 nagios 的相关配置信息进行修改。选择“Configuration → Monitoring Engines → main.cfg”，就可以查看 nagios.cfg 的配置信息，这个文件是 nagios 的主配置文件，记录了 nagios 的配置文件路径、日志文件路径、扩展模块的配置、监控参数配置等信息，可以对这些配置信息进行修改。接着，选择“Configuration → Monitoring Engines → resources”，这是 nagios 的 resource.cfg 文件的配置

信息，主要是设置 nagios 使用插件的路径信息，也可进行修改。最后，还有一个文件 CGI.cfg 用于对 nagios 权限的设置，选择“Configuration → Monitoring Engines → cgi”，即可查看和修改 nagios 相关访问和管理权限。

8.5.4 添加服务和服务组

在完成主机和主机组添加后，下面开始添加需要监控的服务和服务组。在完成 Centreon 安装后，会有一些默认的服务，为了不影响介绍，现将这些默认的监控服务删除，选择“Configuration → Services → Services by host”，如图 8-37 所示。

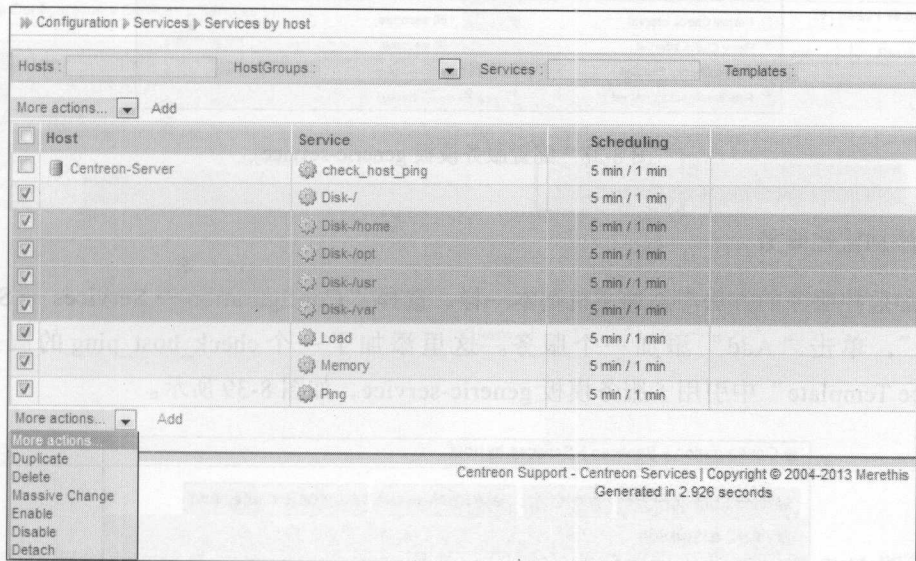


图 8-37 删除默认的服务监控项

1. 配置服务模板 generic-service

这里的服务模板与之前介绍的主机模板类似，它们具有相同的作用，默认的服务模板无需修改即可拿来使用，当然也可以根据需要进行配置修改，在添加服务时引用。

选择“Configuration → Services → Templates”，即可编辑 generic-service 模板，如图 8-38 所示。在这里可以修改或添加“Service State”选项中的属性值。一般情况下，在 generic-service 服务模板中设置的属性值都是通用的或公用的，主要在创建服务的时候进行引用。

Configuration > Services > Templates

Service Configuration Relations Data Processing Service Extended Info

Modify a Service Template Model

General Information

Alias * generic-service

Service Template Name * generic-service

Service Template Model [icon]

Service State

Is volatile ☐ Yes ☒ No ☐ Default

Check Period 24x7

Check Command [icon]

Args

Argument	Value	Example
No argument found for this command		

Max Check Attempts 3

Normal Check Interval 5 * 60 seconds

Retry Check Interval 1 * 60 seconds

Active Checks Enabled ☒ Yes ☐ No ☐ Default

Passive Checks Enabled ☐ Yes ☒ No ☐ Default

图 8-38 配置服务模板 generic-service

2. 添加监控服务

添加监控服务的方法与添加主机基本一样，选择“Configuration → Services → Services by host”，单击“Add”添加一个服务。这里添加了一个 check_host_ping 的服务，在“Service Template”中引用了服务模板 generic-service，如图 8-39 所示。

Configuration > Services > Services by host

Service Configuration Relations Data Processing Service Extended Info

Add a Service

General Information

Description * check_host_ping

Service Template generic-service

Service State

Is Volatile ☐ Yes ☐ No ☒ Default

Check Period *

Check Command * check_ping

Args

Argument	Value	Example
No argument found for this command		

Max Check Attempts *

Normal Check Interval * * 60 seconds

Retry Check Interval * * 60 seconds

Active Checks Enabled ☐ Yes ☐ No ☒ Default

Passive Checks Enabled ☐ Yes ☐ No ☒ Default

图 8-39 添加 check_host_ping 监控服务

在添加监控服务过程中，重点关注“Service State”、“Macros”、“Notification”三个

服务属性，其中“Service State”主要用于设置服务的检查周期、检查的命令、正常检查间隔、重试检查间隔等参数；“Macros”是自定义宏，暂不添加；“Notification”用来设置报警通知属性，可以添加联系人、联系人组、通知间隔、通知周期、通知类型等。关于“Notification”属性，后面会详细介绍。

在完成图 8-39 所示页面的设置后，接着选择“Relations”选项，如图 8-40 所示，这个选项用于选择与此服务相关联的主机，之后在“Linked with Hosts”选项中会列出所有已经添加的主机，选择与此服务相关联的主机即可。最后单击“Save”保存设置，这样一个监控服务就添加完成了。

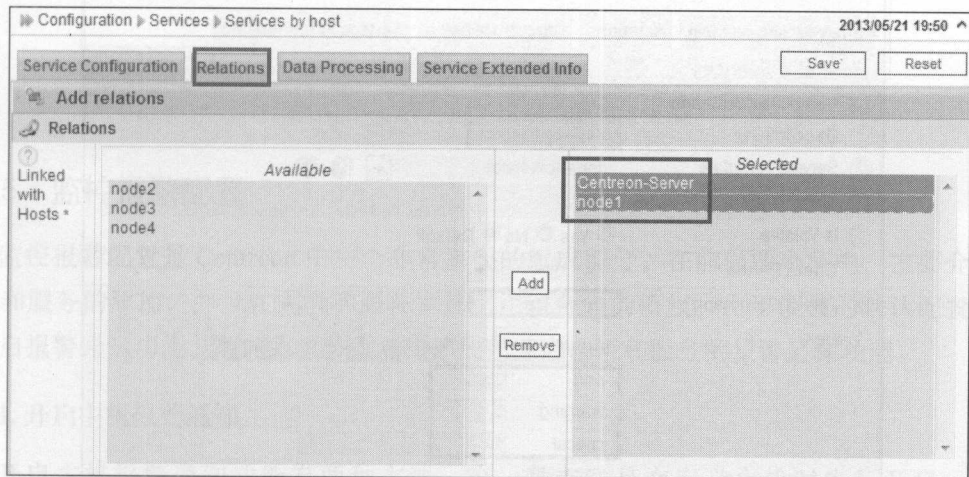


图 8-40 选择与定义服务相关联的主机

可以按照如上方法依次创建多个监控服务，创建完成后的服务列表如图 8-41 所示。

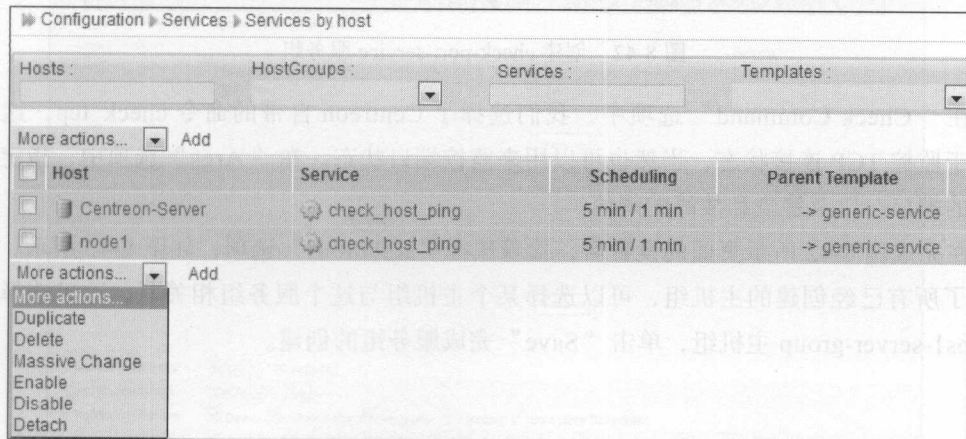


图 8-41 创建的监控服务列表

在服务监控列表中，可以对每个服务进行复制、删除、启用、禁用等操作，这些功能对以后的监控系统运维是非常重要的。

3. 添加监控服务组

选择“Configuration → Services → Services by host group”，单击“Add”按钮添加一个服务组，如图 8-42 所示。与之前添加服务方式一样，这里添加了一个名为“check-port-service”的服务组，并且应用了 generic-service 模板。

Configuration ► Services ► Services by host group																
Service Configuration		Relations	Data Processing	Service Extended Info												
Add a Service																
General Information																
Description *	check-port-service															
Service Template	generic-service															
Service State																
Is Volatile	<input type="radio"/> Yes <input type="radio"/> No <input checked="" type="radio"/> Default															
Check Period *	[Dropdown]															
Check Command *	check_tcp															
Args	<table border="1"> <thead> <tr> <th>Argument</th> <th>Value</th> <th>Example</th> </tr> </thead> <tbody> <tr> <td>port</td> <td>3306</td> <td></td> </tr> <tr> <td>warning</td> <td>5</td> <td></td> </tr> <tr> <td>critical</td> <td>10</td> <td></td> </tr> </tbody> </table>				Argument	Value	Example	port	3306		warning	5		critical	10	
Argument	Value	Example														
port	3306															
warning	5															
critical	10															
Max Check Attempts *	[Input]															
Normal Check Interval *	[Input] * 60 seconds															
Retry Check Interval *	[Input] * 60 seconds															
Active Checks Enabled	<input type="radio"/> Yes <input type="radio"/> No <input checked="" type="radio"/> Default															
Passive Checks Enabled	<input type="radio"/> Yes <input type="radio"/> No <input checked="" type="radio"/> Default															

图 8-42 创建 check-port-service 服务组

在“Check Command”选项中，我们选择了 Centreon 自带的命令 check_tcp，这个命令用于监控 TCP 连接状态，当然也可以用来监控端口状态。在“Args”选项中，指定了要监控的端口，以及警告和故障的阈值。

在完成图 8-42 所示页面的设置后，接着选择“Relations”选项，如图 8-43 所示。这里列出了所有已经创建的主机组，可以选择某个主机组与这个服务组相关联，这里选择的是 Centos1-server-group 主机组，单击“Save”完成服务组的创建。

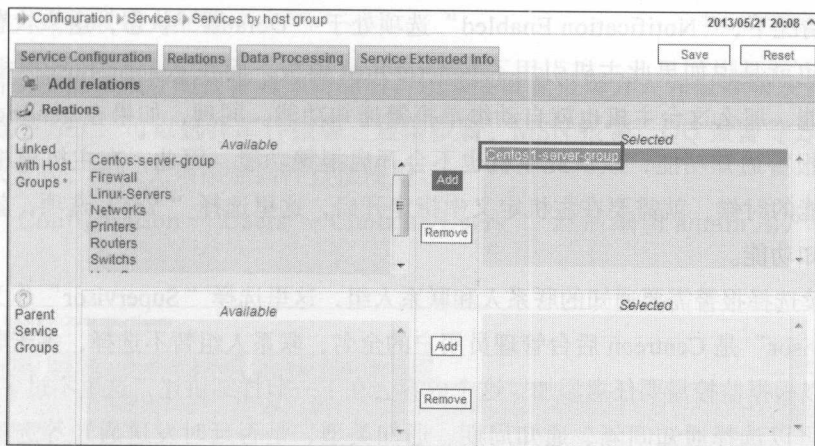


图 8-43 选择与服务组相关联的主机组

8.5.5 监控报警配置

监控报警配置是 Centreon 中一个非常重要的组成部分，在前面两小节中，主要介绍了主机和服务的添加，并且在主机和服务中都引用了各自的模板。由于模板的默认配置并没有开启报警通知功能，因此，本节重点讲述如何开启主机和服务的报警通知功能。

1. 开启主机报警通知

开启主机报警通知功能有两种方法，第一种方法是在定义主机时进行开启，选择“Configuration → Hosts → node1”，编辑已经创建好的主机 node1，这里重点看“Notification”选项，如图 8-44 所示。

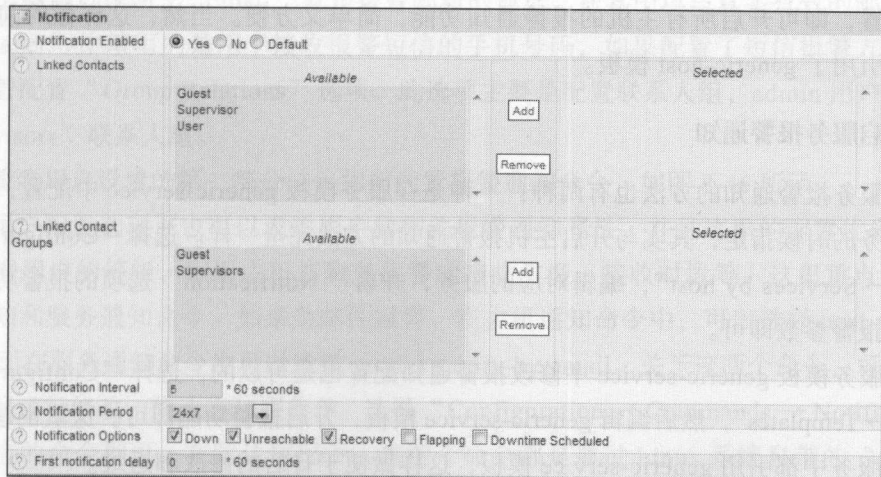


图 8-44 开启主机报警通知功能

在默认情况下,“Notification Enabled”选项处于“Default”状态,这个状态表示一个继承关系,也就是说如果此主机引用了 generic-host 模板,而在 generic-host 模板中开启了报警通知功能,那么这台主机也就自动继承报警通知功能。同理,如果在 generic-host 模板中没有开启报警通知功能,默认此主机也不会开启报警功能,因此,在主机模板没有开启报警通知功能的时候,就需要在主机定义中指定开启,这里选择“Yes”选项,就开启了主机的报警通知功能。

然后继续选择报警需要通知的联系人和联系人组,这里选择“Supervisor”作为通知联系人,“Supervisor”是 Centreon 后台管理员用户的全名,联系人组暂不选择,关于联系人和联系人组,可以根据监控需要任意添加,这个内容会在下一节详细讲述,这里不过多介绍。

接着还可以选择通知间隔、通知周期、通知类型、是否延时发送通知等选项。对于比较重要的主机可以选择短一点的通知时间间隔,可以选择通知周期为 7×24 ,另外还可以选择工作日(workhours)、非工作日(nonworkhours)两个选项,可以选择通知类型为宕机(Down)、不可到达(Unreachable)、恢复(Recovery)等。最后一个选项是设定第一次发送通知的延时时间,如果设置为0,则表示主机故障后立刻发送通知。

开启主机报警的第二种方式是配置主机模板 generic-host。在前面介绍的主机添加过程中,都引用了主机模板 generic-host,而在主机模板配置中也可以开启报警通知功能,开启方法很简单:编辑主机模板,找到“Notification”选项,设置方法与图 8-44 完全一样,当开启了通知功能后,对应的这个主机就继承了模板的设置,自动开启主机通知功能。

根据运维经验,建议通过修改主机模板的方式开启主机通知功能,因为当监控的主机有上千台之多时,一台台修改主机配置变得不现实,此时只须通过修改主机模板 generic-host 的配置,即可开启所有主机的报警通知功能,简单又方便。当然,这样做的前提是所有主机都引用了 generic-host 模板。

2. 开启服务报警通知

开启服务报警通知的方法也有两种:一种是在服务模板 generic-service 中配置,一种是在定义服务的时候指定,其实与开启主机报警通知的方法完全一样。选择“Configuration → Services → Services by host”,编辑对应的服务,开启“Notification”选项的报警功能,并配置各个报警参数即可。

要在服务模板 generic-service 中修改报警通知配置也是可以的,选择“Configuration → Services → Templates”,然后编辑 generic-service 模板,开启报警功能即可。这里也建议将所有添加的服务中都引用 generic-service 模板,这样做便于日后的修改和运维。

3. 报警方式和联系人配置

在开启了主机和服务的报警通知功能后，还需要设置报警方式和报警联系人。Centreon 支持多种报警方式，可以选择邮件报警、短信报警、MSN 报警、QQ 报警等方式，而邮件报警是默认方式，可以无须添加插件直接使用。

选择“Configuration → Users → Contacts/Users”，然后编辑 admin 用户，如图 8-45 所示。

图 8-45 设置 admin 用户报警方式

这是用户设置的第一部分，“Full Name”其实就是 admin 登录用户的全名，在之前的介绍中已经做过说明，“Email”就是报警邮箱的地址，在其中填写真实有效的邮箱地址即可，“Pager”中填写的是用于接收报警短信的手机号码，如果配置了短信报警方式将会用到。接着配置“Group Relations”选项，此选项主要是配置联系人组，admin 用户默认属于“Supervisors”联系人组。

接着看用户设置的第二部分——如何配置报警通知命令，如图 8-46 所示。

在用户配置选项下也可以设置用户的相关报警通知属性，并且这里的配置优先级最高，可以根据用户的等级，配置主机或服务报警的接收类型、接收时段等，这里重点关注一下主机通知和服务通知命令。如果是邮件报警，在主机通知命令中，可以选择 host-notify-by-email，而在服务通知命令中可以选择 service-notify-by-email。关于这两个命令，可能要根据实际情况进行修改，修改方法很简单：选择“Configuration → Commands → Notifications”，编辑对应的命令即可。在默认情况下，这两个命令都是通过 Linux 系统自带的“mail”命令发送邮件的，这个命令不太好用，功能也不强大，因此推荐另一个利用命令行发邮件的

工具“sendEmail”，这个命令行发邮件的工具功能非常强大，使用也非常简单，强烈推荐。下面简单介绍 sendEmail。

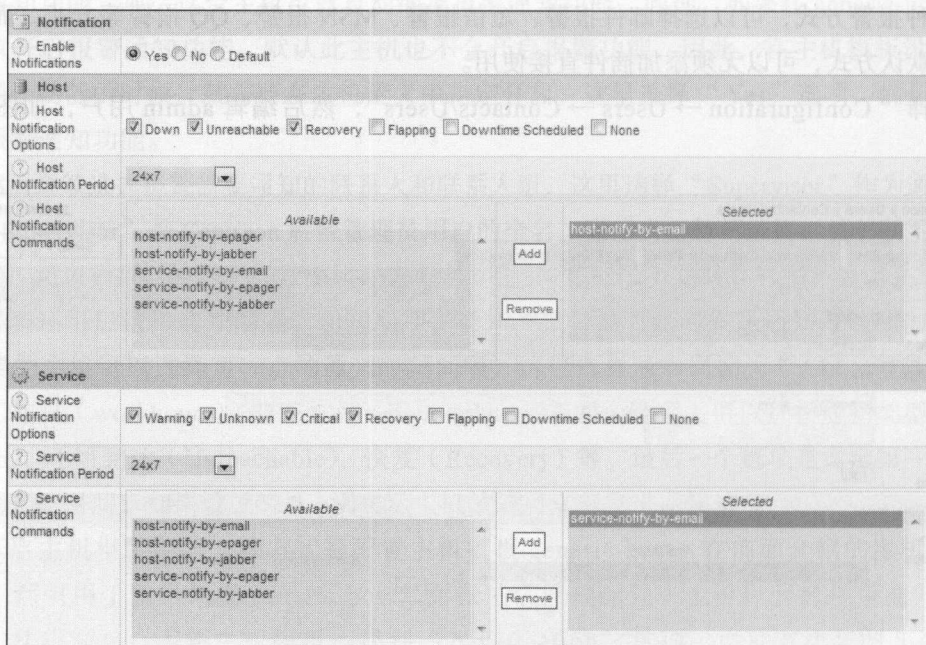


图 8-46 配置 admin 用户报警通知命令

sendEmail 的主页是 <http://caspien.dotconf.net/menu/Software/SendEmail/>，它的安装非常简单，下载下来即可使用。可以将解压出来的 sendEmail 可执行文件复制到 /usr/local/bin 下，直接运行 sendEmail 就会显示详细的用法，这里介绍几个重要的使用参数。

- ❑ -f，表示发送者的邮箱。
- ❑ -t，表示接收者的邮箱。
- ❑ -s，表示 SMTP 服务器的域名或者 IP 地址。
- ❑ -u，表示邮件的主题。
- ❑ -xu，表示 SMTP 验证的用户名。
- ❑ -xp，表示 SMTP 验证的密码。
- ❑ -m，表示邮件的内容。

下面介绍几个简单的例子。

```
cat 文件名 | /usr/local/bin/sendEmail -f centreon@test.com -t admin@ixdba.net -s mail.test.com -u "Centreon host-notify test" -xu centreon -xp 1q2w3e4r
```

这个例子省去了“-m”参数，而是通过管道将邮件内容传给了 sendEmail。下面是一个

定义好的 service-notify-by-email 命令的内容:

```
/usr/bin/printf "%b" "*****centreon Notification *****\n\nNotification
Type: $NOTIFICATIONTYPE$\n\nService: $SERVICEDESC$\nHost: $HOSTALIASE$\n
Address: $HOSTADDRESS$\nState: $SERVICESTATE$\n\nDate/Time: $LONGDATETIME$\n\n
Additional Info:\n\n$SERVICEOUTPUT$" | /usr/local/bin/sendEmail -f centreon@
test.com -t $CONTACTEMAIL$ -s mail.test.com -u "*** $NOTIFICATIONTYPE$ alert -
$HOSTALIASE$/$SERVICEDESC$ is $SERVICESTATE$ ***" -xu centreon -xp 1q2w3e4r
```

到这里为止,关于 Centreon 邮件报警方式的配置基本介绍完成了,下面再介绍一下其他报警方式的优缺点及使用经验。

Centreon 支持多种报警方式,上面介绍了默认的邮件报警,还有通过第三方插件配置短信报警、MSN 或 QQ 在线报警等,但是本章不打算介绍这些方式,主要基于以下原因:

❑ 短信报警、MSN 或 QQ 报警的接入方式与邮件报警的接入方法完全一样,都是通过配置报警命令脚本完成。而短信报警通常都是通过飞信这个免费插件实现的。飞信虽然免费,但是有很大局限性,不适用所有手机运营商,并且及时性和稳定性无法保证,这就会导致出现错报或漏报的问题,而对于一些关键业务系统,这些都是无法容忍的,因此,对于这种情况可以考虑使用商用的短信通道或者购买独立的短信硬件设备,费用并不高,作为公司层面也应该能负担得起。

❑ MSN 或 QQ 在线报警基本都是通过脚本调用监控状态完成报警的,其完成的功能与邮件报警并无区别,因此这种报警方式不推荐使用。

邮件报警是最基础的报警形式,它可以有多种扩展形式,目前很多邮箱都支持短信提醒功能,将邮箱绑定短信提醒后,就变相实现了短信报警的功能。另外,现在的手机基本都是智能机了,在手机上安装一个邮件客户端工具,然后绑定报警邮件地址,就可以随时随地收取报警邮件了,这种方式简单、实用,唯一的成本就是要支付一部分手机的无线通信流量。

4. 查看监控报警状态

Centreon 通过 Web UI 界面展示了所有主机和服务的运行状态,对于不同的运行状态, Centreon 分别用不同的颜色显示,基本的状态与颜色的定义如下:

- ❑ 正常运行状态 (OK), 一般情况下用绿色表示。
- ❑ 警告状态 (Warning), 通常用黄色表示。
- ❑ 故障、宕机状态 (Critical、Down), 通常用红色表示。
- ❑ 未知状态 (Unknown), 通常用灰色表示。
- ❑ 挂起、不可到达状态 (Pending、Unreachable), 通常用浅蓝色表示。

选择“Monitoring → Hosts”即可查看所有主机、主机组、故障主机等的相关信息,如图 8-47 所示为所有主机运行状态。

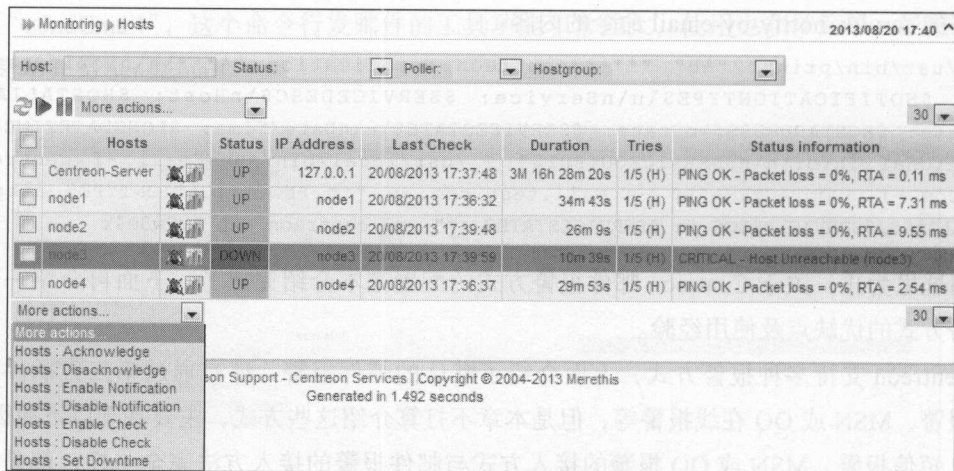


图 8-47 所有主机运行状态

从图 8-47 中可以看出，主机 node3 处于“DOWN”状态，并且此主机也用红色进行了标注。查看最后一列 node3 主机故障的详细信息，发现是“Host Unreachable”状态，此时就需要去检查 node3 主机的网络连接情况了。

从图 8-47 还可以看到所有主机的主机名称、主机运行状态、主机 IP 地址、最后一次检查的时间、某种主机状态持续的时间等，最后一列是主机状态的检查结果。在“Hosts”一列后面的那个小图标代表这些主机没有启用主机报警，在启用主机报警后，这个小图标会自动消失。这个小图标后面的图形报表是对 ping 操作监控状态的一个数据汇总，单击这个图标即可查看主机在某段时间内的 ping 状态曲线图。

Centreon 查看主机或服务状态的功能非常强大，支持根据主机名过滤查询，还支持根据状态查询、根据主机组名称查询等。另外还可以对主机进行启用通知、关闭通知、启用检查、关闭检查等操作。

选择“Monitoring → Services → All Services”即可查看所有监控服务的状态，如图 8-48 所示。

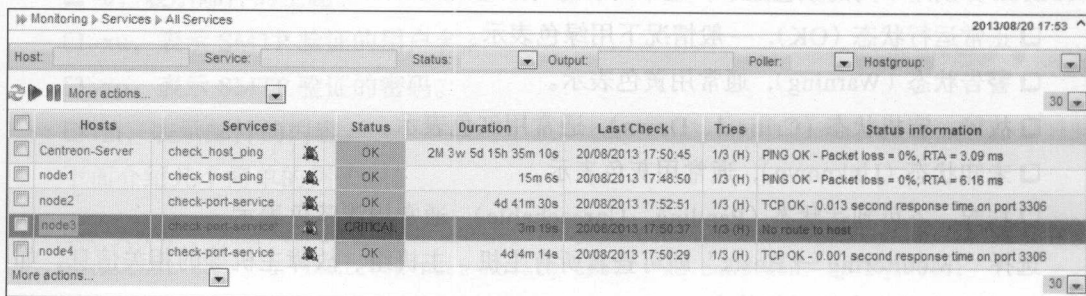
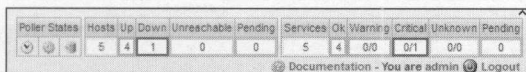


图 8-48 所有监控服务运行状态

从图 8-48 中可以看出, 由于 node3 主机网络故障无法连接导致 node3 上的 MySQL 端口监控服务出现故障, 具体的监控出错信息是 “No route to host”, 这种错误一般都是网络无法连接导致的。

关于对监控状态的查看, Centreon 还给出了一个全局的状态统计, Centreon 后台右上角的一个表格统计了所有主机和服务的运行状态, 如图 8-49 所示。



Poller States	Hosts Up	Down	Unreachable	Pending	Services Ok	Warning	Critical	Unknown	Pending
	5	4	1	0	5	4	0/0	0/1	0/0

Documentation - You are admin Logout

图 8-49 主机和服务的全局状态监控

在 Centreon 监控系统中, 所有监控状态页面都是定时自动刷新的, 这个刷新值可以自行修改, 默认刷新闻隔是一分钟, 所以只需要查看全局的这个状态监控表格, 就能知道哪些主机或服务出现了故障。

8.5.6 用户和用户权限管理

Centreon 是一个多用户、多角色管理系统, 从功能上可以将用户划分为两个部分: 一部分是后台登录用户, 另一部分是报警通知用户。后台登录用户主要用于对主机或服务的添加、修改和状态查看等。结合 Centreon 的用户权限管理功能, 可以针对不同用途的用户设置不同的登录级别, 也就是说可以授权某个用户管理一批主机和服务, 而没有授权的其他主机和服务, 此用户将无法查看和管理, 通过这种方式, Centreon 实现了对主机和服务的分级、分权限管理。

Centreon 的另一部分用户主要用于报警通知, 其实就是在 nagios 中所说的联系人和联系人组, 在添加用户时, 会要求输入用户的邮件地址、电话号码等, 在发送报警通知时, 如果指定了某个联系人, 那么会自动调用此联系人的邮件地址或电话号码进行邮件报警或短信提醒, 如果有很多用户都需要接收某类报警邮件时, 那么可以将这些有相同需求的用户都添加到一个组中, 这就是联系人组。

1. 添加用户

下面演示如何添加一个用户, 以及如何向用户授权。

选择 “Configuration → Users → Contacts/Users”, 单击 “Add” 按钮添加一个用户, 如图 8-50 所示。

这里假定添加一个 ixdba 用户, 在 “Full Name” 中填写 ixdba, 在 “Alias / Login” 中同样填写 ixdba, 然后添加 “Email” 地址为 ixdba@ixdba.net, “Linked to Contact Groups” 暂不填写, 然后启用 ixdba 的报警通知功能。

Configuration > Users > Contacts / Users 2013/08/21 12:03

General Information Centreon Authentication Additional Information Save Reset

Add a User

General Information

Full Name * ixdba

Alias / Login * ixdba

Email * ixdba@ixdba.net

Pager

Contact template used

Group Relations

Linked to Contact Groups

Available: Guest, Supervisors

Selected:

Add Remove

Notification

Enable Notifications ☒ Yes ☐ No ☐ Default

图 8-50 添加用户 ixdba

然后选择 ixdba 用户接收报警的类型、时段和报警命令，如图 8-51 所示。

Host

Host Notification Options ☒ Down ☒ Unreachable ☒ Recovery ☐ Flapping ☐ Downtime Scheduled ☐ None

Host Notification Period 24x7

Host Notification Commands

Available: host-notify-by-epager, host-notify-by-jabber, service-notify-by-email, service-notify-by-epager, service-notify-by-jabber

Selected: host-notify-by-email

Add Remove

Service

Service Notification Options ☒ Warning ☒ Unknown ☒ Critical ☒ Recovery ☐ Flapping ☐ Downtime Scheduled ☐ None

Service Notification Period 24x7

Service Notification Commands

Available: host-notify-by-email, host-notify-by-epager, host-notify-by-jabber, service-notify-by-epager, service-notify-by-jabber

Selected: service-notify-by-email

Add Remove

图 8-51 设置 ixdba 用户接收报警的类型、时段和报警命令

根据用户的类别和要求选择相应的报警属性即可，然后选择“Centreon Authentication”标签，填写 ixdba 用户登录 Centreon 后台的密码、语言等信息，如图 8-52 所示。

“Centreon Authentication”选项是可选的，如果要开启 ixdba 用户登录 Centreon 平台的功能，就需要设置登录密码等信息，如果添加的用户仅仅是为了接收报警通知，那么无须设置登录密码。在这个标签中还可以设置此用户是否为管理员、登录的默认语言等信息，

设置完成，单击“Save”按钮，完成用户添加。

Configuration ► Users ► Contacts / Users 2013/08/21 12:14

General Information Centreon Authentication Additional Information Save Reset

Add a User

Centreon

Reach Centreon Front-end * ☒ Yes ☐ No

Password Generate

Confirm Password

Default Language * en_US

Admin * ☐ Yes ☒ No

Autologin Key Generate

Authentication Source * Centreon

Access lists

Access list groups

Available		Selected
ALL	Add Remove	

图 8-52 设置 ixdba 用户登录 Centreon 后台的密码

2. 添加用户组

选择“Configuration → Users → Contact Groups”，单击“Add”添加一个用户组，如图 8-53 所示。

Add a Contact Group

General Information

Contact Group Name * View_group

Alias * View

Relations

Linked Contacts

Available		Selected
Guest Supervisor User	Add Remove	ixdba

Additional Information

Status ☒ Enabled ☐ Disabled

图 8-53 添加用户组 View_group

这里添加一个名为 View_group 的用户组，然后在“Linked Contacts”项将刚才创建的用户 ixdba 加入此用户组。

3. 添加访问组

下面添加一个访问组，选择“Administration → ACL → Access Groups”，可以看到有个

默认访问组 ALL，单击“Add”添加一个新的访问组，如图 8-54 所示。

图 8-54 添加访问组 access_group

这里添加一个名为 access_group 的访问组，然后将 ixdba 用户加入这个访问组中，当然也可以直接将之前创建的 View_group 用户组加入 access_group 访问组中。

4. 添加访问控制权限

下面开始创建菜单的访问控制权限，选择“Administration → ACL → Menus Access”，可以看到默认四个菜单访问控制项，分别是 Configuration、Graphs、Monitoring+Home 和 Reporting，这里根据需要再新增一个菜单访问控制项，单击“Add”添加一个“Menus Access”，如图 8-55 所示。

这里添加一个名为 guest_view 的菜单访问权限，从图 10-55 可以看出，此权限只能访问 Centreon 的 Home、Monitoring、Views 和 Reporting 菜单，而对于 Configuration 和 Administration 菜单将无法访问，其实就是只有查看和预览权限，没有添加和修改权限。最后将这个菜单访问权限授予 access_group 访问组，这样 access_group 访问组下的所有用户将只有查看和预览的权限。

5. 添加资源访问策略

最后一步，还需要添加资源访问策略。资源访问策略用于控制用户对主机或服务访问的控制。选择“Administration → ACL → Resources Access”，默认有个“All Resources”资源访问策略，单击“Add”添加一个新的资源访问策略，如图 8-56 所示。

Add an ACL

General Information

ACL Definition *

Alias

Status ☒ Enabled ☐ Disabled

Linked Groups

Available: ALL

Selected: access_group

Add Remove

Accessible Pages

+ Home : ☒

+ Monitoring : ☒

+ Views : ☒

+ Reporting : ☒

+ Configuration : ☐

+ Administration : ☐

图 8-55 增加一个菜单访问控制权限

Add an ACL

General Information Hosts Resources Services Resources Meta Services Filters

Save Delete

General Information

Access list name *

Description

People linked to this Access list

Linked Groups

Available: ALL

Selected: access_group

Add Remove

图 8-56 添加资源访问策略

这里添加一个名为“view_acl”的访问策略，并且将 access_group 访问组加入这个访问策略中，接着选择“Hosts Resources”标签，选择可访问的主机资源，如图 8-57 所示。

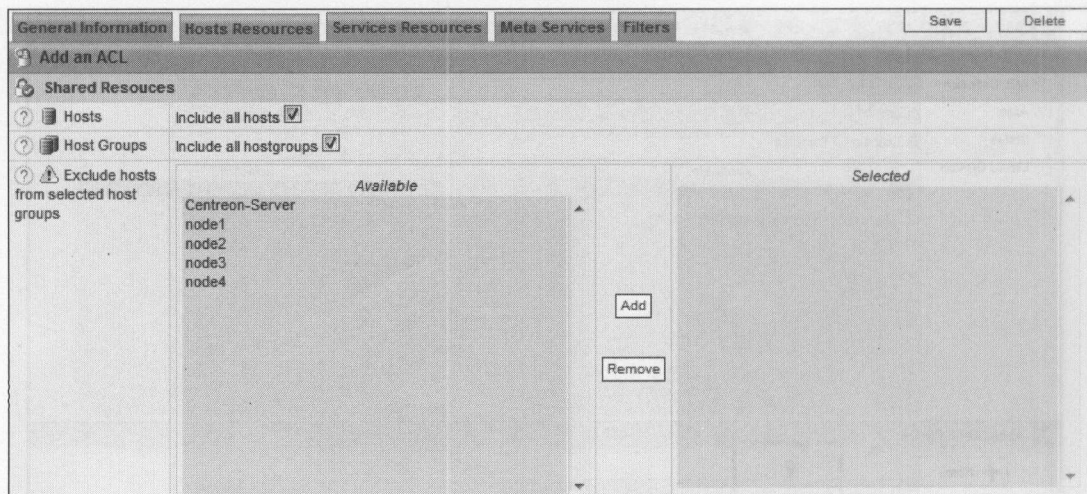


图 8-57 选择可访问的主机资源

这里选择所有的主机和主机组，当然也可以选择指定的一批主机或主机组，继续选择“Services Resources”标签，选择可访问的服务组，如图 8-58 所示。

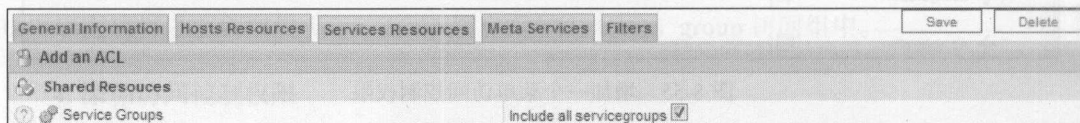


图 8-58 选择可访问的服务组

这里选择所有服务组，设置完成后单击“Save”保存，这样就成功添加了一个资源访问策略。

在完成所有配置策略添加后，就可以通过 ixdba 用户登录进入 Centreon 后台了，此时可以看到 ixdba 用户只能看到 Home、Monitoring、Views 和 Reporting 四个菜单选项，并且能看到所有主机和服务的运行状态，这表明资源控制策略生效了。

8.6 配置分布式监控

支持分布式监控是 Centreon 系统核心的功能，在前面章节简单介绍了 Centreon 的分布式监控架构，本节将对分布式监控进行详细阐述。

8.6.1 分布式监控架构与实现原理

在介绍分布式监控之前，先介绍一下我们要实现的分布式监控的架构，如图 8-59 所示。

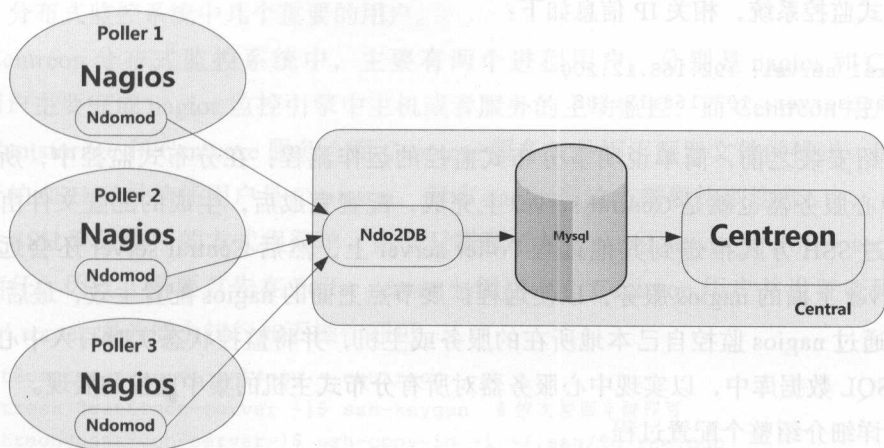


图 8-59 典型的 Centreon 分布式监控架构

从图 8-59 可以看出，Centreon 的分布式监控是由多个 Poller 和一个 Central 构成的。在 Centreon 分布式监控中，远程的扩展节点称为 Poller，而本地的中心服务器称为 Central，在通常情况下，远程的 Poller 服务器上仅需要安装 nagios 监控引擎及 ndoutils 模块，而 Central 服务器上需要安装 nagios、nagios-plugins、ndoutils、MySQL、Centreon 和 Centreon Web 等，这些前面章节已经做过详细介绍。

远端的 Poller 服务器通过自身安装的 ndomod 模块与 Central 服务器上的 Ndo2DB 进行通信，最终将监控状态写入 MySQL 数据库，以供 Centreon 调用。

8.6.2 分布式监控搭建环境介绍

下面我们将通过一个实例详细介绍下 Centreon 分布式监控的搭建过程，搭建环境如图 8-60 所示。

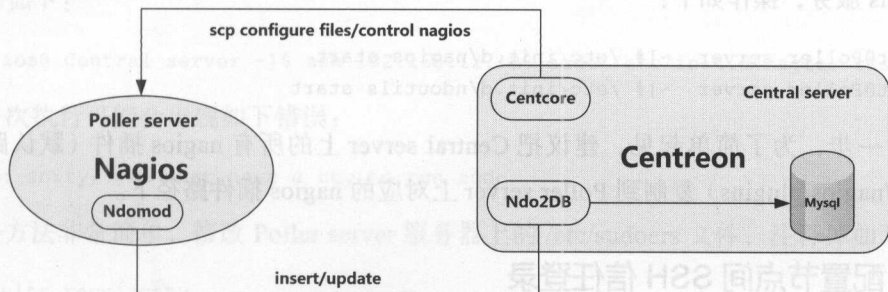


图 8-60 两个节点的 Centreon 分布式监控

本实例环境是一个中心服务器（Central server）和一个远程扩展节点（Poller server）组

成的分布式监控系统，相关 IP 信息如下：

```
Central server: 192.168.12.200
Poller server: 192.168.12.188
```

在介绍安装之前，简单说明下分布式监控的运作流程：在分布式监控中，所有的配置都在中心服务器也就是 Central server 上完成，配置完成后，生成的配置文件由 Central server 通过 SSH 方式推送到其他远程 Poller server 上，然后 Central server 还会远程重启 Poller server 上面的 nagios 服务，以使远程扩展节点上面的 nagios 配置生效，最后，Poller server 会通过 nagios 监控自己本地所在的服务或主机，并将监控状态实时写入中心服务器上的 MySQL 数据库中，以实现中心服务器对所有分布式主机的集中监控和展现。

下面详细介绍整个配置过程。

8.6.3 监控软件的安装

监控软件的安装分为两个部分：首先是 Central server 端的安装，主要安装有 nagios、nagios-plugins、ndoutils、nrpe、Centreon 等软件，安装方法前面章节已经介绍，这里不再讲述；最后是 Poller server 端的安装，在 Poller server 端的安装非常简单，仅需要安装 nagios 和 ndoutils 两个部分，如果需要 nrpe 服务，还可以安装 nrpe，安装方法前面也都已介绍，这里也不再多讲。

在 Poller server 端安装完 nagios 后，默认的配置文件的目录为 /etc/nagios，以后从 Central server 上推送过来的配置文件，默认也会放到这个路径下，因此要保证 /etc/nagios 目录对 Central server 有读写权限，需要在 Poller server 上做如下授权：

```
[root@Poller server ~]# chown -R centreon:centreon /etc/nagios
```

为什么要做这个授权，8.6.4 节有详细介绍。接着，还需要在 Poller server 上启动 nagios 和 ndoutils 服务，操作如下：

```
[root@Poller server ~]# /etc/init.d/nagios start
[root@Poller server ~]# /etc/init.d/ndoutils start
```

最后一步，为了简单起见，建议把 Central server 上的所有 nagios 插件（默认路径为 /usr/lib64/nagios/plugins）复制到 Poller server 上对应的 nagios 插件路径下。

8.6.4 配置节点间 SSH 信任登录

为了让 Central server 上生成的配置文件能自动分发到 Poller server 上，需要在 Central server 和 Poller server 之间进行 SSH 单向信任，在进行用户信任登录之前，需要了解下

Centreon 分布式监控系统中几个重要的用户。

在 Centreon 分布式监控系统中，主要有两个进程用户，分别是 nagios 和 Centreon，nagios 用户主要完成 nagios 监控引擎中主机或者服务的主动监控，而 Centreon 用户主要用来管理 Centstorage 和 Centcore 服务，而 Centcore 服务就是用于配置文件的推送，因此，这里要执行的 SSH 登录信任用户是 Centreon，而非 nagios，这点需要特别注意。

配置 SSH 登录信任的方式很简单，由于只需配置从 Central server 到 Poller server 之间的单向信任，因此，需要首先在 Poller server 上创建一个 Centreon 用户并设置密码，然后在 Central server 服务器上执行如下操作即可：

```
[root@centreon-server /]# su - centreon
[centreon@centreon-server ~]$ ssh-keygen # 依次按回车键即可
[centreon@centreon-server~]$ ssh-copy-id -i ~/.ssh/id_rsa.pub
centreon@192.168.12.188 # 根据提示，输入 192.168.12.188 服务器 Centreon 用户的密码即可
[centreon@centreon-server~]$ ssh 192.168.12.188 date # 验证 SSH 信任登录是否成功
```

接下来，为了能让 Poller server 上的 Centreon 用户管理 nagios 服务，还需要在 Poller server 上进行 sudo 配置，其实也就是在 /etc/sudoer 文件的最后添加如下内容：

```
centreon ALL=NOPASSWD: /etc/init.d/nagios restart
centreon ALL=NOPASSWD: /etc/init.d/nagios stop
centreon ALL=NOPASSWD: /etc/init.d/nagios start
centreon ALL=NOPASSWD: /etc/init.d/nagios reload
centreon ALL=NOPASSWD: /usr/bin/nagiosstats
centreon ALL=NOPASSWD: /usr/bin/nagios *
centreon ALL=NOPASSWD: /etc/init.d/ndoutils *
```

这样配置以后，Central server 上面的 Centreon 用户就可以通过无密码的方式登录到 Poller server 上远程管理 nagios 服务，实现 Central server 对 Poller server 的分布式远程监控。

修改完成后，在 Central server 的 Centreon 用户下执行 ssh sudo 命令来测试设置是否正确，操作如下：

```
[nagios@ Central server ~]$ ssh 192.168.12.188 sudo /etc/init.d/nagios restart
```

第一次执行可能会出现如下错误：

```
sudo: sorry, you must have a tty to run sudo
```

解决方法非常简单，修改 Poller server 服务器上的 /etc/sudoers 文件，注释掉如下属性：

```
Defaults requiretty
```

再次执行上面的命令，应该就能顺利执行了。

8.6.5 在 Central server 上添加分布式监控配置

1. 添加一个 Poller

选择“Configuration → Centreon → Pollers”，单击“Add”添加一个新的 Poller，为了方便起见，也可以直接复制一个默认的 Poller，然后进行简单修改即可。图 8-61 展示的是复制的默认名为 Central 的 Poller。

Modify a poller Configuration	
Server Information	
Poller Name *	myCentral
IP Address	192.168.12.188
Localhost ?	<input type="radio"/> Yes <input checked="" type="radio"/> No
Is default poller ?	<input type="radio"/> Yes <input checked="" type="radio"/> No
SSH Information	
SSH port	22
Monitoring Engine Information	
Engine	Nagios
Monitoring Engine Init Script	/etc/init.d/nagios
Monitoring Engine Binary	/usr/bin/nagios
Monitoring Engine Statistics Binary	/usr/bin/nagiosstats
Perfdata file	/var/log/nagios/service-perfdata

图 8-61 添加一个名为 myCentral 的 Poller

这里重点关注三个选项：“Poller Name”是指定一个 Poller 名称，这里是“myCentral”，可以随意指定；“IP Address”就是 Poller server 的 IP 地址，这里输入“192.168.12.188”；最后一个是“SSH port”，默认是 22 端口，可根据情况修改。其他配置无须改动，在配置完成后，保存退出即可。

2. 为 Poller 添加 ndomod 配置

ndomod 配置就是指定 Poller server 所要连接到 Central server 的信息，包括 Central server 的 IP 地址、通信端口等设置。

选择“Configuration → Centreon → ndomod.cfg”，单击“Add”添加 myCentral 的 ndomod.cfg 配置。同理，为了方便，可以直接复制默认的 ndomod.cfg 配置 Central-mod，修改后的结果如图 8-62 所示。

这里重点关注方框中的 5 个选项：在“Description”中输入的“myndomod”是添加的 ndomod 配置的名称；在“Instance Name”中选择刚刚添加的名为 myCentral 的 Poller；将“Status”标记为“Enabled”状态；在“Interface Type”中一定要选择“tcpsocket”；

最后一个“Output”选项非常重要，它表示将监控状态输出到哪个节点，这里一定要填写 Central server 的 IP 地址，即为“192.168.12.200”。后面还有几个选项，按照默认的配置即可，无须修改。

Modify a ndomod Configuration File	
Description	
Description *	myndomod
Instance Name	myCentral
Status	<input checked="" type="radio"/> Enabled <input type="radio"/> Disabled
Output	
Interface Type	tcpsocket
Output	192.168.12.200
Buffer File	
TCP Port	5668
Buffer size of the interface	5000
Rotations	
Rotation interval	14400
Rotation command	
Rotation timeout	60
Database Disconnexions	
Reconnection interval	15
Notification interval in case of disconnection	900

图 8-62 添加 myCentral 的 ndomod.cfg 配置

3. 为 Poller 添加 ndo2db 配置

ndo2db 服务只在 Central server 上存在，选择“Configuration → Centreon → ndo2db.cfg”单击“Add”添加 myCentral 的 ndo2db.cfg 配置，这里同样直接复制默认的 ndo2db.cfg 配置 Principal，修改后的结果如图 8-63 所示。

Modify a ndo2db Configuration File	
Ndo2db Informations	
Description *	myndo2db
Status	<input checked="" type="radio"/> Enabled <input type="radio"/> Disabled
Requester	myCentral
Socket Type	tcp
Socket Name	/var/run/ndo.sock
TCP Port	5668
Ndo2db execution access	
User ndo2db	nagios
Group ndo2db	centreon
<div>Save Reset</div>	

图 8-63 添加 myCentral 的 ndo2db.cfg 配置

在“General”标签中, 需要注意的选项有“Requester”, 这里选择“myCentral”即可, “Socket Type”选择“tcp”, 而“TCP Port”只要和 ndomod.cfg 配置中对应选项保持一致即可, 默认是 5668, 这个端口在 Central server 上由 Ndo2db 程序启动, 远程各个 Poller server 通过该程序和 Central server 上的 MySQL 数据库交互存取监控数据。

接着看“Database”标签, 如图 8-64 所示。

Database Information for ndo2db	
Database Type	MySQL
Database Hoster	192.168.12.200
Database Name	centreon_status
Listening Port	3306
Prefix	nagios_
User	centreon
Password	*****

图 8-64 配置 ndo2db 的数据库连接信息

本页面要配置的是远程各个 Poller server 要连接的数据库信息, 这里选择 MySQL 数据库, 然后提供 MySQL 数据库所在的服务器 IP 地址、数据库名、连接端口、表名称前缀、连接数据库的用户名和密码等信息。由于是远程连接 MySQL, 因此需要在数据库中进行授权, 以保证 Poller server 可以正常连接到 MySQL 数据库中。

注意, 这里的“Database Hoster”一定要填写 MySQL 数据库的 IP 地址, 如果 MySQL 数据库和 Centreon 在一台机器上, 不建议填写“localhost”或“127.0.0.1”之类的地址。

授权的方法比较简单, 登录 MySQL 所在服务器的 SQL 命令行, 执行类似如下操作:

```
mysql> grant all ON centreon_status.* to centreon@'129.168.12.188' identified by 'xxxxxx';
mysql> flush privileges;
```

4. 为 Poller 添加 nagios 主配置文件

选择“Configuration → Monitoring Engines → main.cfg”, 单击“Add”添加 myCentral 的 main.cfg 配置, 这里复制默认的 main.cfg 配置“Nagios CFG 1”, 修改后的结果如图 8-65 所示。

这里需要注意的选项有“Configuration Name”和“Linked poller”, “Configuration Name”用来指定配置文件名称, 这里定义为“MyNagiosCFG1”, “Linked poller”用于指定适用的 poller, 这里选择“myCentral”即可。其他选项无须任何修改, 保持默认即可。

图 8-65 为 myCentral 添加 nagios 主配置文件

至此，在 Centreon 上添加分布式监控的 Web 配置已经介绍完毕。

5. 添加 Poller 后端主机

在成功添加一个 Poller server 后，接下来为这个 Poller server 添加一批监控主机，添加监控主机的方法与直接介绍过的添加主机过程完全相同，不过，需要修改一个配置，如图 8-66 所示。

图 8-66 为 Poller server 添加一个后端主机

这里添加了一个 node5 主机，IP 为 192.168.12.100，同样继承了 generic-host 主机模板，需要注意的是，这里的“Monitored from”选项选择“myCentral”，其他设置与之前介绍的添加主机过程完全一致，不再介绍。

6. 重载 Centreon 生成配置文件

在所有分布式监控配置完成之后，需要执行重载 Centreon，生成分布式配置文件，选择“Configuration → Monitoring Engines → Generate”，如图 8-67 所示。

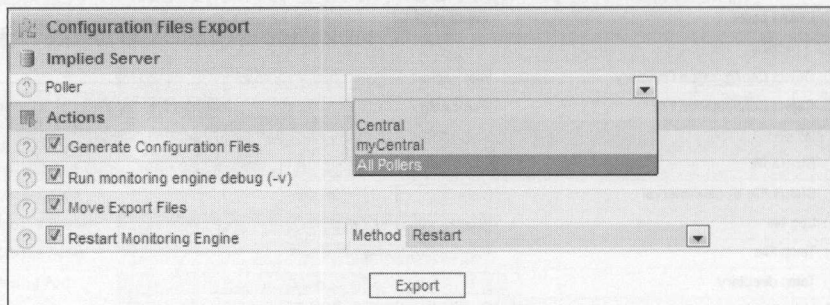


图 8-67 生成分布式配置文件

在“Poller”选项中选择“All Pollers”，然后选择所有“Actions”，执行“Restart”操作，生成所有 Poller 的配置文件。在执行完这个操作后，可以登录到 Poller server 上，看看配置文件是否已经正常推送到 /etc/nagios 目录下，并查看 nagios 进程是否被重新启动过，如果是，那么分布式监控系统运行正常。

选择“Configuration → Centreon → Pollers”，查看两个 Poller 的运行状态，如图 8-68 所示。

More actions... Add									
<input type="checkbox"/>	Name	IP Address	Localhost	Is running ?	Conf Changed *	PID	Start time	Last Update	Version
<input checked="" type="checkbox"/>	Central	127.0.0.1	Yes	Yes	No	5953	31/08/2013 23:10:12	07/09/2013 18:22:13	Nagios 3.2.3
<input checked="" type="checkbox"/>	myCentral	192.168.12.188	-	Yes	No	13483	07/09/2013 17:42:56	07/09/2013 18:22:13	Nagios 3.2.3
More actions... Add									

图 8-68 Centreon 多 Poller 运行状态

从图 8-68 可以看出，两个 Poller 都处于正常运行状态，并且还能看到每个 Poller 对应的 IP 地址，以及 nagios 引擎的启动时间、更新时间、运行 PID 以及 nagios 版本信息等。

至此，基于 nagios 的 Centreon 分布式监控系统配置完毕。

8.7 常见服务监控配置

在前面的内容中，只是简单介绍了 Centreon 的几个基础监控项，例如添加主机、服务等，这些对于一个监控系统来说是远远不够的，本节将重点介绍对一些扩展服务的监控，

也就是对一些常见应用的监控。

8.7.1 nagios 插件编写规范

作为一名运维人员，相信大家对脚本并不陌生，而 nagios 插件就是可执行的一个脚本或程序，这个脚本可以用各种语言实现，例如 shell、Perl、Python 都可以。对 nagios 熟悉的读者肯定也编写过不少 nagios 插件，虽然 nagios 自带了很多常用的监控插件，但是由于监控环境不同，运维人员往往根据需求编写适合自己的 nagios 插件。这里先简单介绍下编写 nagios 插件的一些规则。

nagios 插件必须完成以下两个功能：

❑ 脚本执行结果必须返回几个可能返回值中的一个。

❑ 脚本执行结果至少返回一行文本输出。

nagios 插件作为一个向 nagios 传递状态的工具，它并不关心程序的具体执行细节，读者可以根据自己的需要去监控 Web 服务、数据库服务、网络服务和磁盘空间等，但是在监控脚本的输出结果中必须包含以下返回值之一，因为 nagios 就是靠这些返回值来判断监控服务运行状态的。

nagios 插件的返回值如表 8-1 所示。

表 8-1 nagios 插件的返回值

返回值	服务状态	主机状态
0	OK	UP
1	WARNING	UP or DOWN/UNREACHABLE
2	CRITICAL	DOWN/UNREACHABLE
3	UNKNOWN	DOWN/UNREACHABLE

另外，nagios 脚本在输出结果时也是有规范的，基本要求是插件至少返回一行文本，从 nagios 3 开始，插件可以返回多行文本数据了，这样可以输出更多有用的信息供监控查阅。

8.7.2 监控 Apache 运行状态

1. 获取 Apache 监控脚本

在介绍监控 Apache 的状态之前，需要一个监控 Apache 状态的插件，这里并不打算介绍插件如何编写，因为已经有很多编写好的插件供 nagios 使用。在 nagios 的官方网站 <http://exchange.nagios.org/directory/Plugins> 上有很多可供直接使用的插件分类，找到 Web Servers 分类，其中就有许多对 Apache 的监控脚本，这里选取一个以 Python 语言编写的 Apache 监控脚本，下载地址是：<http://exchange.nagios.org/directory/Plugins/Web-Servers/>

Apache/check_apache2-2Epy/details, 下载脚本名称为 check_apache2.py, 然后将此脚本放到监控服务器上的 /usr/lib64/nagios/plugins 目录下, 并授予可执行权限:

```
[root@centreon-server ~]# chmod 755 /usr/lib64/nagios/plugins/check_apache2.py
```

2. 修改被监控服务器 Apache 配置

监控 Apache 是通过 Apache 的 mod_status 模块实现的, 因此需要在被监控的机器上打开 mod_status 模板, 基本方法就是修改 Apache 的配置文件, 添加如下内容:

```
<Location /server-status>
SetHandler server-status
Order deny,allow
Deny from all
Allow from 127.0.0.1 192.168.12.200 192.168.12.188
</Location>
ExtendedStatus On
```

在上面的配置中“Allow from”表示允许访问“/server-status”页面的主机 IP, 将监控主机 IP 加入即可, 如果有多个主机需要访问这个页面, 以空格分开, 依次添加 IP 地址。

3. 在监控服务器上配置 Apache 监控

在监控服务器的 Centreon Web 界面, 选择“Configuration → Commands → Checks”, 然后单击“Add”新建一个 Command, 如图 8-69 所示。

图 8-69 新建一个名为 check_apache_status 的 Command

在图 8-69 中, 重点看“Command Line”的内容, 其中,“\$USER1\$”就是监控服务器

上 nagios 监控插件的路径，这里是“/usr/lib64/nagios/plugins”，之前已经把 Apache 的监控脚本 check_apache2.py 放到了这个路径下，这里直接引用即可。在这个脚本对应的参数中，定义了三个参数变量“\$ARG1\$”、“\$ARG2\$”和“\$ARG3\$”，分别用于指定 Apache server 的端口、警告阈值和故障阈值，这里建议将每个参数变量的作用都在“Argument Descriptions”选项中做个描述，这样在添加服务时就不容易出错。

接着开始添加监控 Apache 的服务，选择“Configuration → Services → Services by host”，单击“Add”添加一个服务，如图 8-70 所示。

The screenshot shows the 'Add a Service' form in Nagios. The 'Service Configuration' tab is active. The form includes the following fields and sections:

- Add a Service**: A button to initiate the service creation.
- General Information**:
 - Description ***: check_apache
 - Service Template**: generic-service (selected from a dropdown)
- Service State**:
 - Is Volatile**: Radio buttons for Yes, No, and Default (selected).
 - Check Period ***: 24x7 (selected from a dropdown)
 - Check Command ***: check_apache_status (selected from a dropdown)
 - Args**: A table defining arguments for the check command.

Argument	Value	Example
Port, Default is: 80	80	
WARNING, Default is: -1	200	
CRITICAL, Default is: -2	500	
 - Max Check Attempts ***: (empty field)
 - Normal Check Interval ***: 30 * 60 seconds
 - Retry Check Interval ***: (empty field) * 60 seconds
 - Active Checks Enabled**: Radio buttons for Yes, No, and Default (selected).
 - Passive Checks Enabled**: Radio buttons for Yes, No, and Default (selected).

图 8-70 添加 check_apache 服务

添加服务的方法，前面已经做过介绍，首先看“Service Configuration”标签中的几个重要选项，“Description”就是监控服务的名称，这里定义为 check_apache。“Service Template”是指定服务的模板，这里仍然选择“generic-service”。下面重点关注下“Check Command”选项，这个选项用来指定服务检查的命令，这里选择刚才创建的命令 check_apache_status。最后一个选项“Args”就是刚才创建 check_apache_status 命令时指定的三个变量，可以看到，由于在定义命令时增加了每个变量的含义，因此这里在添加服务时就能非常清楚地知道每个变量是什么含义，根据每个变量的含义，依次填写 Apache 状态页面的端口号、报警阈值和故障阈值。

接着看“Relations”标签，如图 8-71 所示。

这个标签主要是指定要监控的 Apache 主机，在图 8-71 左边的主机列表中选择要监控的主机即可，这里选择 node2 主机。在完成所有设置后，单击“Save”保存退出，check_apache 服务就添加完成了。

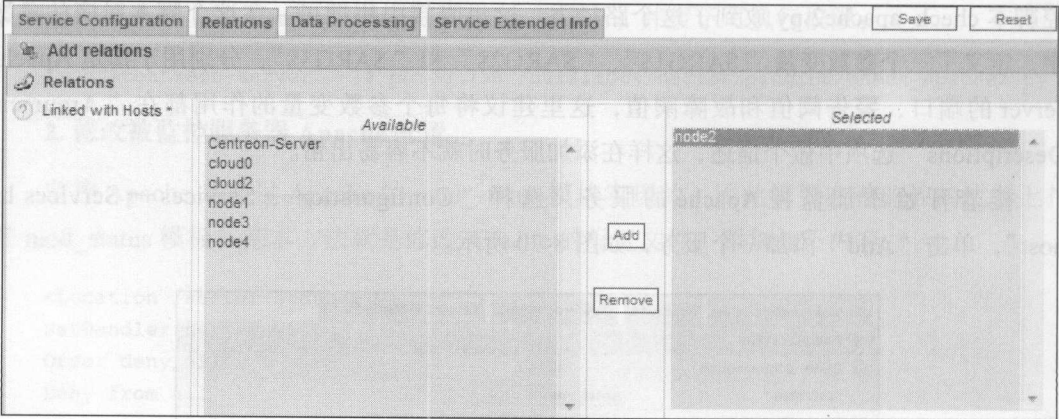


图 8-71 指定需要监控的 Apache 主机

4. 加载配置使添加服务生效

在服务添加完成后，打开“Configuration → Monitoring Engines → Generate”，选择对应的 Poller，然后重启 nagios 监控引擎，查看新添加的 check_apache 服务运行状态，如图 8-72 所示。

Hosts	Services	Status	Duration	Last Check	Tries	Status information
Centreon-Server	check_host_ping	OK	5d 9h 43m 24s	11/09/2013 18:30:41	1/3 (H)	PING OK - Packet loss = 0%, RTA = 3.71 ms
cloud0	check_ping	OK	1w 3d 4h 15m 57s	11/09/2013 18:29:55	1/3 (H)	PING OK - Packet loss = 0%, RTA = 19.60 ms
cloud2	check_ping162	OK	1w 3d 4h 22m 24s	11/09/2013 18:30:32	1/3 (H)	PING OK - Packet loss = 0%, RTA = 23.57 ms
node1	check_host_ping	OK	56m 11s	11/09/2013 18:31:47	1/3 (H)	PING OK - Packet loss = 0%, RTA = 19.62 ms
node2	check-port-service	OK	3w 4d 23h 1m 7s	11/09/2013 18:30:59	1/3 (H)	TCP OK - 0.051 second response time on port 3306
	check_apache	OK	7h 2m 10s	11/09/2013 18:29:48	1/3 (H)	OK: Apache serves 0.001729 requests per second. 1 busy workers, 7 idle workers
node3	check-port-service	OK	1d 26m 22s	11/09/2013 18:30:59	1/3 (H)	TCP OK - 0.021 second response time on port 3306
node4	check-port-service	OK	1w 5d 8h 35m 59s	11/09/2013 18:30:51	1/3 (H)	TCP OK - 0.000 second response time on port 3306

图 8-72 check_apache 服务运行状态

可以看到，check_apache 服务的状态是“OK”，并且在最后给出了服务的状态信息。至此，check_apache 服务添加完成。

8.7.3 监控 MySQL 运行状态

现在基于 MySQL 的应用越来越多，因此对 MySQL 数据库的监控也成为运维工作中必不可少的一部分。对 MySQL 的监控可以有多种形式，可以监控 MySQL 的运行状态，具体包含端口、运行负载、库 / 表状态等；也可以专门监控 MySQL 的查询状态，当查询负荷达到指定的阈值时进行报警。本节重点介绍如何监控 MySQL 的运行状态。

在添加 MySQL 监控之前，仍然需要编写对 MySQL 监控的插件，不过幸运的是，

nagios 自带的插件中已经包含了对 MySQL 监控的插件，可以在 nagios 插件默认安装路径 /usr/lib64/nagios/plugins 找到，对应的插件名称为 check_mysql。要了解这个脚本如何使用，可通过 “/usr/lib64/nagios/plugins/check_mysql -h” 获取使用帮助，最后，还需要保证这个脚本有可执行权限。接下来介绍 MySQL 监控配置阶段。

1. 在被监控的 MySQL 上添加远程可访问的用户

由于 check_mysql 这个脚本需要登录到远程的 MySQL 服务器上获取运行状态，因此，需要在被监控的 MySQL 服务器上创建一个可供监控服务器访问的用户。这里创建一个名为 nagios 的 MySQL 用户，基本操作如下：

```
mysql> grant usage ON *.* to nagios@'%' identified by 'xxxxxx';
mysql> flush privileges;
```

2. 在监控服务器上添加监控 MySQL 状态的命令

在监控服务器的 Centreon Web 界面，选择 “Configuration → Commands → Checks”，然后单击 “Add” 新建一个 Command，如图 8-73 所示。

The screenshot shows the 'Modify a Command' form in the Centreon Web interface. The form is titled 'Check' and contains the following fields and options:

- Command Name:** check_mysql_status
- Command Type:** Notification (radio button), Check (radio button, selected), Misc (radio button), Discovery (radio button)
- Command Line:**
 - Text area: \$USER1\$/check_mysql -H \$HOSTADDRESS\$ -P \$ARG1\$ -u \$ARG2\$ -p \$ARG3\$
 - Buttons: << (left), >> (right)
 - Variables: \$USER1\$ (path to the plugins), /check_mysql, \$ADMINEMAILS
- Enable shell:** ☐
- Argument Example:** \$HOSTADDRESS\$
- Argument Descriptions:**
 - Describe arguments (button)
 - Clear arguments (button)
 - ARG1: Mysql Port, default: 3306
 - ARG2: Connect Mysql username
 - ARG3: connection password

图 8-73 添加一个名为 check_mysql_status 的 Command

check_mysql 脚本有多个参数选项，这里仅使用四个参数，“-H”表示主机名，“-P”指定被监控的 MySQL 数据库监听端口，“-u”表示连接远程 MySQL 数据库的用户名，“-p”表示连接 MySQL 的密码。设置完成，单击 “Save” 保存，check_mysql_status 命令创建完成。

接着开始添加 MySQL 监控服务，选择 “Configuration → Services → Services by host”，单击 “Add” 添加一个服务，如图 8-74 所示。

The screenshot shows the 'Service Configuration' tab for adding a new service named 'check_mysql'. The configuration is as follows:

Field	Value												
Description *	check_mysql												
Service Template	generic-service												
Service State	Service is enabled												
Is Volatile	<input type="radio"/> Yes <input checked="" type="radio"/> No <input type="radio"/> Default												
Check Period *	24x7												
Check Command *	check_mysql_status												
Args	<table border="1"> <thead> <tr> <th>Argument</th> <th>Value</th> <th>Example</th> </tr> </thead> <tbody> <tr> <td>Mysql Port, default: 3306</td> <td>3308</td> <td></td> </tr> <tr> <td>Connect Mysql username</td> <td>nagios</td> <td></td> </tr> <tr> <td>connection password</td> <td>*****</td> <td></td> </tr> </tbody> </table>	Argument	Value	Example	Mysql Port, default: 3306	3308		Connect Mysql username	nagios		connection password	*****	
Argument	Value	Example											
Mysql Port, default: 3306	3308												
Connect Mysql username	nagios												
connection password	*****												
Max Check Attempts *	3												
Normal Check Interval *	30 * 60 seconds												
Retry Check Interval *	30 * 60 seconds												
Active Checks Enabled	<input type="radio"/> Yes <input checked="" type="radio"/> No <input type="radio"/> Default												
Passive Checks Enabled	<input type="radio"/> Yes <input checked="" type="radio"/> No <input type="radio"/> Default												

图 8-74 添加 check_mysql 服务

首先看“Service Configuration”标签，与之前添加服务方式相同，在“Check Command”选项中选择刚刚创建的命令“check_mysql_status”，然后在“Args”选项中依次输入MySQL的监控端口为3308，以及连接MySQL的用户名和密码。

接着查看“Relations”标签，如图8-75所示，选择check_mysql服务要监控的主机，这里选择“cloud0”，最后，单击“Save”保存退出即可完成服务的添加。

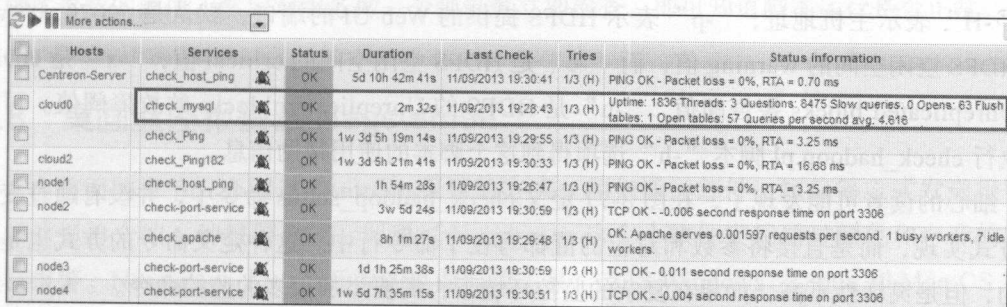
The screenshot shows the 'Relations' tab where hosts are assigned to the service. The 'Available' list contains: Centreon-Server, cloud2, node1, node2, node3, and node4. The 'Selected' list contains: cloud0. The 'Add' button is visible between the two lists.

图 8-75 选择 check_mysql 服务要监控的主机

3. 加载配置使监控服务生效

打开“Configuration → Monitoring Engines → Generate”，选择对应的 Poller，然后重

启 nagios 监控引擎，查看新添加的 check_mysql 服务运行状态，如图 8-76 所示。



Hosts	Services	Status	Duration	Last Check	Tries	Status information
Centreon-Server	check_host_ping	OK	5d 10h 42m 41s	11/09/2013 19:30:41	1/3 (H)	PING OK - Packet loss = 0%, RTA = 0.70 ms
cloud0	check_mysql	OK	2m 32s	11/09/2013 19:28:43	1/3 (H)	Uptime: 1836 Threads: 3 Questions: 8475 Slow queries: 0 Opens: 63 Flush tables: 1 Open tables: 57 Queries per second avg: 4.816
cloud2	check_ping	OK	1w 3d 5h 19m 14s	11/09/2013 19:29:55	1/3 (H)	PING OK - Packet loss = 0%, RTA = 3.25 ms
node1	check_host_ping	OK	1h 54m 28s	11/09/2013 19:26:47	1/3 (H)	PING OK - Packet loss = 0%, RTA = 3.25 ms
node2	check-port-service	OK	3w 5d 24s	11/09/2013 19:30:59	1/3 (H)	TCP OK - 0.006 second response time on port 3306
node3	check_apache	OK	8h 1m 27s	11/09/2013 19:29:48	1/3 (H)	OK: Apache serves 0.001597 requests per second. 1 busy workers, 7 idle workers.
node4	check-port-service	OK	1d 1h 25m 38s	11/09/2013 19:30:59	1/3 (H)	TCP OK - 0.011 second response time on port 3306
node4	check-port-service	OK	1w 5d 7h 35m 15s	11/09/2013 19:30:51	1/3 (H)	TCP OK - 0.004 second response time on port 3306

图 8-76 check_mysql 运行状态

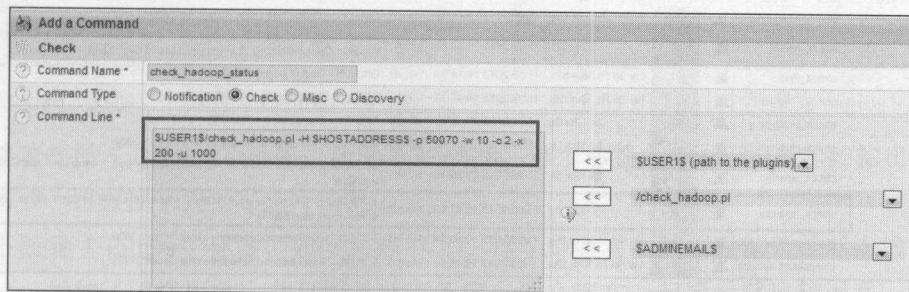
从图 10-76 可以看出，MySQL 服务状态为“OK”，并给出了服务状态的详细信息。为了验证监控的有效性，可以停止 cloud0 上的 MySQL 服务，看监控系统是否能在下个监控周期发现 MySQL 的异常状态。

8.7.4 监控 Hadoop HDFS 运行状态

对 Hadoop 的状态监控是云平台运维人员必不可少的工作，由于 Hadoop 提供了 Web UI 访问界面可以查看 HDFS 的状态，因此，可以借助于这个 Web UI 界面对 HDFS 进行监控，那么，首先还是要编写监控 HDFS 的脚本。为了减少工作量，这里还是套用 nagios 官方网站的一个成熟脚本，可以从 http://exchange.nagios.org/directory/Plugins/Java-Applications-and-Servers/check_hadoop_hdfs/details 下载这个基于 Perl 编写的 HDFS 监控脚本，将下载的脚本命名为 check_hadoop.pl，放到监控服务器上的 /usr/lib64/nagios/plugins 目录下，并授予可执行权限。下面进入 HDFS 监控配置阶段。

1. 在监控服务器添加监控 HDFS 状态的命令

在监控服务器的 Centreon Web 界面选择“Configuration → Commands → Checks”，然后单击“Add”新建一个 Command，如图 8-77 所示。



Add a Command

Check

Command Name: check_hadoop_status

Command Type: ☐ Notification ☒ Check ☐ Misc ☐ Discovery

Command Line: `$USER1$/check_hadoop.pl -H $HOSTADDRESS$ -p 50070 -w 10 -c 2 -x 200 -u 1000`

Path to the plugins: \$USER1\$ (path to the plugins)

check_hadoop.pl

\$ADMINEMAILS

图 8-77 添加一个名为 check_hadoop_status 的 Command

在创建 `check_hadoop_status` 命令过程中，使用了 `check_hadoop.pl` 脚本的 6 个参数，其中“-H”表示主机地址，“-p”表示 HDFS 提供的 Web UI 的端口，默认是 50070，“-w”是 HDFS 空闲空间的 warning 值，而“-c”是 HDFS 空闲空间的 critical 值，“-x”是 HDFS 的 Unreplicated blocks 警告阈值，“-u”是 HDFS 的 Unreplicated blocks 的故障阈值。可通过执行 `check_hadoop.pl` 脚本“-h”选项得到这个脚本的使用帮助信息。

细心的读者可能发现了，在图 8-77 定义 `check_hadoop_status` 命令中，并没有通过变量的方式实现，而是直接将参数和对应的值都写在了命令行中。这种定义命令的方式也是可以的，但是灵活性不高，如果在复杂的监控环境下，需要添加很多的监控命令。

接着，开始添加 `check_hadoop` 服务，如图 8-78 所示，添加方法已经介绍过很多遍，这里不再细述。

The screenshot shows the 'Add a Service' dialog in Nagios. The 'General Information' tab is active. The 'Description' field contains 'check_hadoop'. The 'Service Template' is set to 'generic-service'. Under 'Service State', 'Is Volatile' is set to 'Default'. 'Check Period' is '24x7'. 'Check Command' is 'check_hadoop_status'. The 'Args' section shows a table with columns 'Argument', 'Value', and 'Example', with the message 'No argument found for this command'. At the bottom, 'Max Check Attempts' is 3, 'Normal Check Interval' is 30 seconds, 'Retry Check Interval' is 60 seconds, and both 'Active Checks Enabled' and 'Passive Checks Enabled' are set to 'Default'.

图 8-78 添加 `check_hadoop` 服务

2. 查看 `check_hadoop` 服务运行状态

在完成服务添加后，仍然需要加载配置以使添加的服务生效，然后查看服务状态页面，如图 8-79 所示。

Hosts	Services	Status	Duration	Last Check	Tries	Status information
Centreon-Server	check_host_ping	OK	5d 11h 19s	11/09/2013 19:45:41	1/3 (H)	PING OK - Packet loss = 0%, RTA = 0.30 ms
cloud0	check_mysql	OK	20m 10s	11/09/2013 19:28:43	1/3 (H)	Uptime: 1836 Threads: 3 Questions: 8475 Slow queries: 0 Opens: 63 Flush tables: 1 Open tables: 57 Queries per second avg: 4.616
cloud2	check_ping	OK	1w 3d 5h 36m 52s	11/09/2013 19:44:55	1/3 (H)	PING OK - Packet loss = 0%, RTA = 16.71 ms
cloud2	check_ping182	OK	1w 3d 5h 39m 19s	11/09/2013 19:48:33	1/3 (H)	PING OK - Packet loss = 0%, RTA = 32.13 ms
node1	check_host_ping	OK	2h 12m 6s	11/09/2013 19:46:47	1/3 (H)	PING OK - Packet loss = 0%, RTA = 14.28 ms
node2	check-port-service	OK	3w 5d 18m 2s	11/09/2013 19:45:59	1/3 (H)	TCP OK - 0.000 second response time on port 3306
	check_apache	OK	8h 19m 5s	11/09/2013 19:29:48	1/3 (H)	OK: Apache serves 0.001597 requests per second. 1 busy workers, 7 idle workers.
	check_hadoop	OK	1m 51s	11/09/2013 19:47:02	1/3 (H)	2013-09-11 - OK: HDFS is OK at 65 % free space with no dead nodes, and 140 Under-replicated blocks
node3	check-port-service	OK	1d 1h 43m 16s	11/09/2013 19:45:59	1/3 (H)	TCP OK - 0.046 second response time on port 3306
node4	check-port-service	OK	1w 5d 7h 52m 53s	11/09/2013 19:45:51	1/3 (H)	TCP OK - 0.006 second response time on port 3306

图 8-79 查看 `check_hadoop` 服务状态

至此，check_hadoop 服务添加完成，如果要验证监控脚本的有效性，可以尝试修改监控脚本参数的阈值，然后对服务做一个强制性立即检查，即可知道脚本运行是否正常。

8.8 桌面监控报警器 Nagstamon

在前面的章节，介绍过 Centreon 报警的方式，例如通过邮件、短信等方式发送报警信息，这里再介绍一种报警方式，即通过 Nagstamon 这个小软件实现通过颜色和声音的方式进行报警。Nagstamon 是一个开源的桌面监控工具，支持 Windows、Linux 和 Mac OS X 平台，通过 Nagstamon 可以监控 nagios 运行状态。同时 Nagstamon 还支持多种监控引擎，常见的有 Icinga、Opsview、Centreon、Op5 Monitor/Ninja、Check_MK Multisite 等。更详细的信息可以参考 Nagstamon 官网：<http://nagstamon.ifw-dresden.de/>。这里下载一个 Windows 下的版本 Nagstamon 0.9.9，安装完成后做一个简单的配置即可。

打开 Nagstamon，选择“Servers”标签，单击“New server”按钮添加一个监控，如图 8-80 所示。这里添加一个类型为 Centreon 的监控，在“Monitor URL”中输入 Centreon 监控服务器的 Web 访问地址即可，然后依次输入登录 Centreon 后台的用户名和密码，单击“OK”保存退出。

如果仅仅是对 nagios 进行监控，可以选择“Type”为 Nagios，然后依次输入“Monitor URL”和“Monitor CGI URL”，如图 8-81 所示。这里重点关注下两个 URL 地址，最后输入登录 nagios Web 的用户名和密码即可。

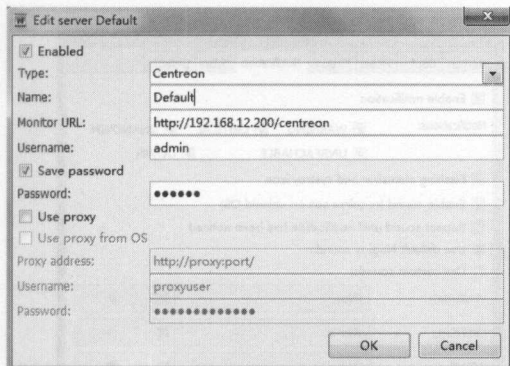


图 8-80 添加一个类型为 Centreon 的监控

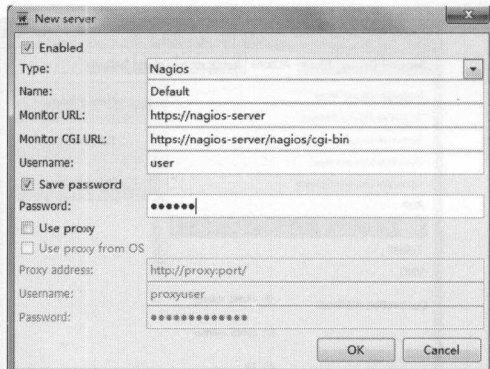


图 8-81 添加一个类型为 nagios 的监控

接着选择“Display”标签，如图 8-82 所示，这个标签主要用于设置监控输出到桌面的展示方式，可根据个人喜好进行设定。

下面看“Filters”标签，如图 8-83 所示，这个标签主要用于过滤监控状态输出。如果 Centreon 添加了很多服务，并且服务状态不定，那么都在一起展示就比较混乱，例如有些

服务没有开启通知机制，有些服务没有开启检查机制，这些显然是不需要监控的，此时，就可以在这里设置过滤机制，将过滤掉这些不需要监控的服务。

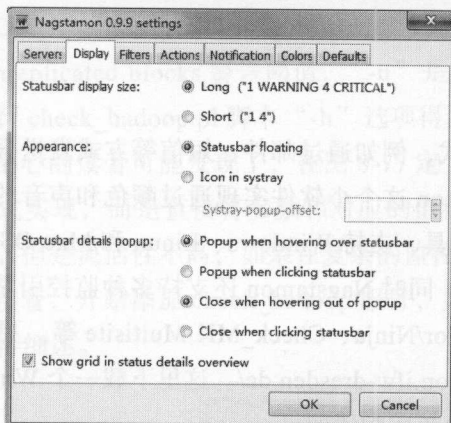


图 8-82 设置监控输出到桌面的展示方式

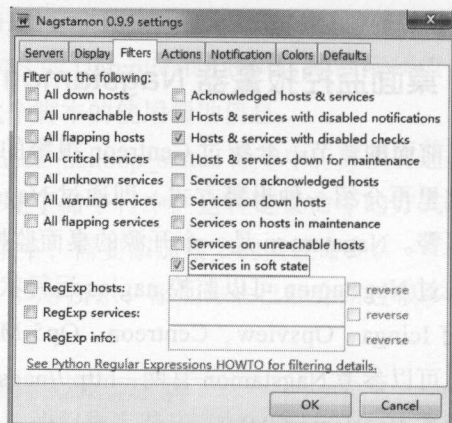


图 8-83 过滤不需要监控的选项

接着看“Actions”标签，如图 8-84 所示，这个标签主要用于定义一些连接服务器的工具，例如 SSH、Telnet、NVC 等，通过定义这些工具，可以在某台服务器出现故障时快速连接。

下一个标签是“Notification”，如图 8-85 所示，这个标签主要用于定义故障通知机制，例如发生什么类型的故障才进行通知，以及检测到故障后针对不同故障级别报警声音的定义等。系统默认有一套故障报警声音，如果觉得不好，还可以自己添加不同报警级别的声音。

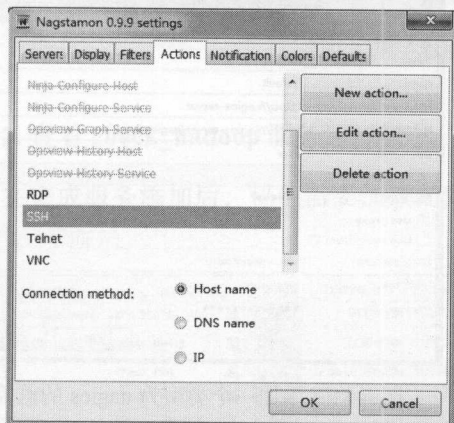


图 8-84 定义快速连接服务器工具

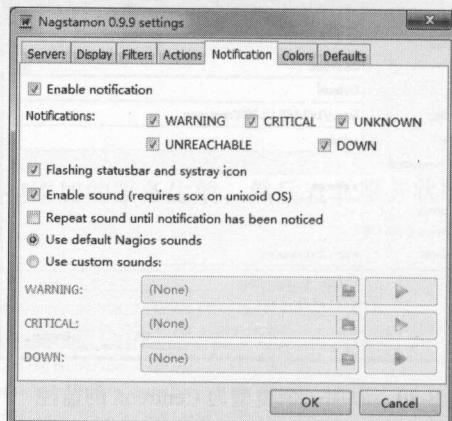


图 8-85 定义报警通知类型和报警声音

最后一个标签是“Colors”，如图 8-86 所示，这个标签用于设置监控状态的显示颜色，在默认情况下，OK 状态显示绿色，WARNING 状态显示黄色，CRITICAL 状态显示红色，UNKNOWN 状态显示橙色，而 UNREACHABLE 状态显示棕色，DOWN 状态则以黑色表示。

如果觉得默认状态的颜色显示不够明确,还可以自己定义各种状态对应的颜色。

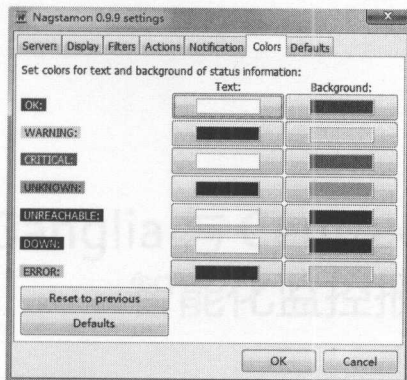


图 8-86 设置监控状态的显示颜色

在完成所有设置后, Nagstamon 会自动查询各台主机和服务的状态, 如果主机或服务出现故障, 监控状态条就会在桌面屏幕上不停闪动, 同时发出报警声音, 在鼠标放到状态条上后, 显示报警状态的详细信息, 如图 8-87 所示。

nagstamon 0.9.9									
Choose monitor...									
admin@Default Monitor Hosts Services History Status: Connected									
Host	Service	Status	Last Check	Duration	Attempt	Status	Information		
node1		DOWN	12/09/2013 14:15:36	58s	1/5		(Host Check Timed Out)		
node4		DOWN	12/09/2013 14:13:37	4h 15m 52s	5/5		CRITICAL - Host Unreachable (node4)		
node4	check_port-service	CRITICAL	12/09/2013 14:16:38	4h 14m 50s	3/3		No route to host		
node2	check_hadoop	WARNING	12/09/2013 14:16:03	42s	1/3		2013-09-12 - WARNING: Under-replicated blocks are more than warning limit 100 %		

图 8-87 Nagstamon 监控展示的报警信息

通过 Ganglia 与 Centreon 构建 智能化监控报警平台

9.1 智能运维监控报警平台的组成

随着大数据时代的来临，运维工作的难度越来越大，每名运维人员都要面临不计其数的服务器和海量的数据，如何保证众多服务器和业务系统稳定高效地运行并尽量减少死机时间，成为考核运维工作的重要指标，而要实现大规模的运维，必须有一套行之有效的智能运维监控管理系统，本章就详细介绍如何构建一套完善的运维监控报警平台。

运维的核心工作可以分为运行监控和故障处理两个方面，对业务系统进行精确、完善的监控，保证能够在第一时间发现故障并迅速通知运维人员处理故障是运维监控系统要实现的基础功能；一个功能完善的智能监控系统，不但可以自动处理一些简单故障，减少运维工作量，还应该应用可能出现故障时预先发出报警，预防故障发生。因此，构建一个智能的运维监控平台，必须以运行监控和故障报警这两个方面为重点，将所有业务系统中涉及的网络资源、硬件资源、软件资源、数据库资源等纳入统一的运维监控平台中，并通过消除管理软件的差别、数据采集手段的差别，对各种不同的数据来源实现统一管理、统一规范、统一处理、统一展现、统一用户登录、统一权限控制，最终实现运维规范化、自动化、智能化的大运维管理。

一个智能的运维监控平台一般的设计架构从低到高可以分为 6 层，如图 9-1 所示。

□ 数据收集层：位于第一层，即最底层，主要收集网络数据、业务系统数据、数据库数据、操作系统数据等，然后将收集的数据进行规范化并存储。

- 数据展示层：位于第二层，是一个 Web 展示界面，主要是将数据收集层获取的数据进行统一展示，展示的方式可以是曲线图、柱状图、饼状态等，通过将数据图形化，可以帮助运维人员了解一段时间内主机或网络的运行状态和运行趋势，并作为运维人员排查问题或解决问题的依据。
- 数据提取层：位于第三层，主要是对从数据收集层获取的数据进行规格化和过滤处理，提取需要的数据到监控报警模块，这个部分是监控和报警两个模块的衔接点。
- 报警规则配置层：位于第四层，主要是根据第三层获取到的数据进行报警规则设置、报警阈值设置、报警联系人设置和报警方式设置等。
- 报警事件生成层：位于第五层，主要是对报警事件进行实时记录，将报警结果存入数据库以备调用，并将报警结果形成分析报表，以统计一段时间内的故障率和故障发生趋势。
- 用户展示管理层：位于最顶层，是一个 Web 展示界面，主要是将监控统计结果、报警故障结果进行统一展示，并实现多用户、多权限管理，实现统一用户和统一权限控制。

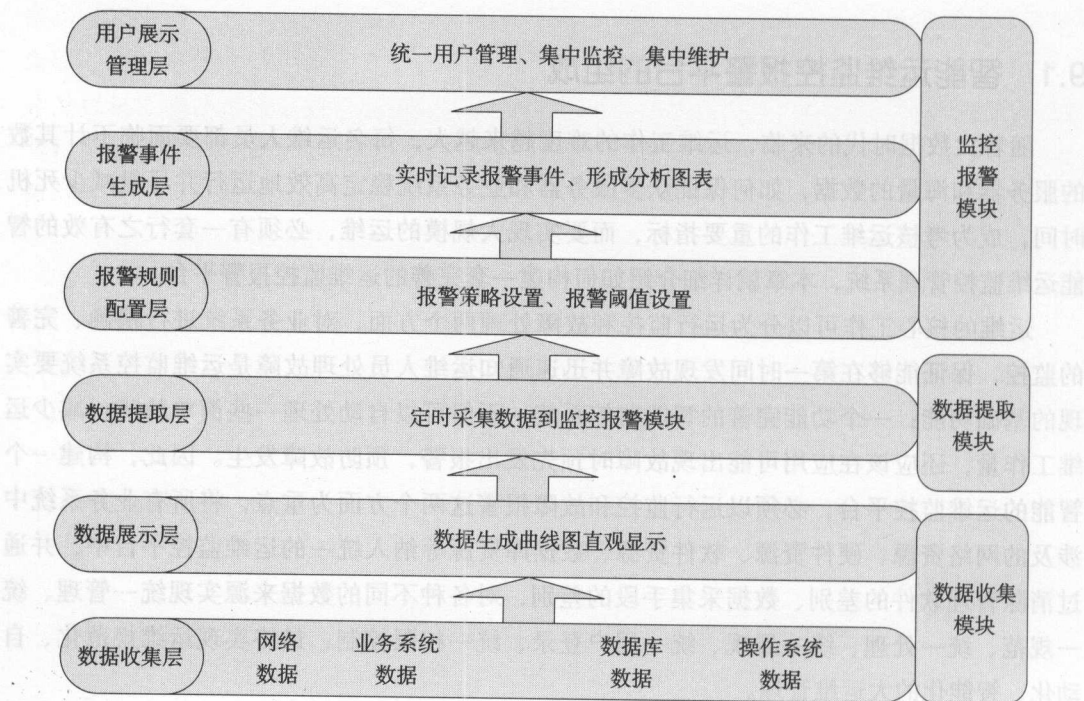


图 9-1 智能运维监控平台的一般设计架构

在这 6 层中，从功能实现划分，又分为三个模块，分别是数据收集模块、数据提取模

块和监控报警模块，每个模块完成的功能如下。

- ❑ 数据收集模块：此模块主要完成基础数据的收集与图形展示。数据收集的方式有很多种，可以通过 SNMP 实现，也可以通过代理模块实现，还可以通过自定义脚本实现。常用的数据收集工具有 Cacti、Ganglia 等。
- ❑ 数据提取模块：此模板主要完成数据的筛选过滤和采集，将需要的数据从数据收集模块提取到监控报警模块中。可以通过数据收集模块提供的接口或自定义脚本实现数据的提取。
- ❑ 监控报警模块：此模块主要完成监控脚本的设置、报警规则设置、报警阈值设置、报警联系人设置等，并将报警结果进行集中展现和历史记录。常见的监控报警工具有 nagios、Centreon 等。

在了解了运维监控平台的一般设计思路之后，接下来详细介绍如何通过软件实现这样一个智能运维监控系统。

图 9-2 是根据图 9-1 的设计思路形成的一个运维监控平台实现拓扑图，从图中可以看出，主要由三大部分组成，分别是数据收集模块、监控报警模块和数据提取模块，其中，数据提取模块用于其他两个模块之间的数据通信，而数据收集模块可以由一台或多台数据收集服务器组成，每个数据收集服务器可以直接从服务器群组收集各种数据指标，经过规范数据格式，最终将数据存储到数据收集服务器中。监控报警模块通过数据提取模块从数据收集服务器获取需要的数据，然后设置报警阈值、报警联系人等，最终实现实时报警。报警方式支持手机短信报警、邮件报警等，另外，也可以通过插件或者自定义脚本来扩展报警方式。这样一整套监控报警平台就基本实现了。

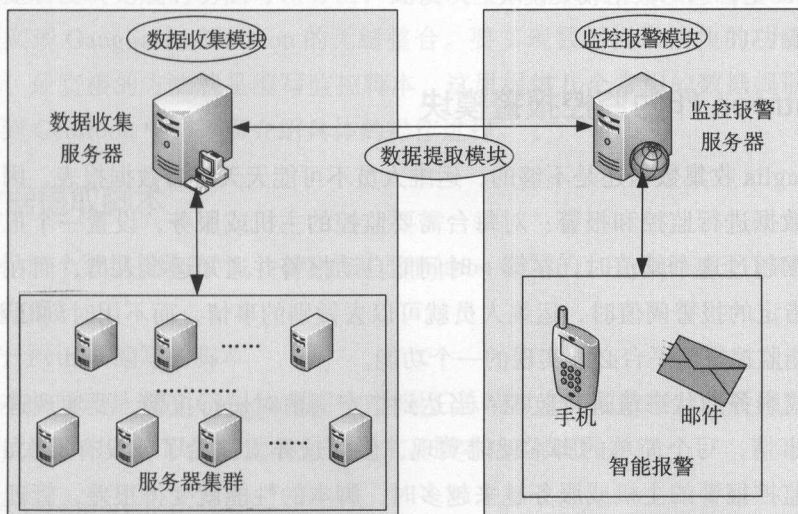


图 9-2 常见运维监控平台拓扑结构

9.2 Ganglia 作为数据收集模块

关于 Ganglia 的基本应用,在前面章节已经详细介绍过,这里将 Ganglia 作为监控报警平台的数据收集模块,主要基于以下几方面的原因:

1) 灵活的分布式、分层体系结构,使 Ganglia 支持上万个监控节点的数据收集,并且性能表现稳定,同时, Ganglia 也可以根据地域环境、网络结构的不同,分地域、分层次灵活部署 Ganglia 数据收集点,而对于数据收集节点可以动态添加或删除,对 Ganglia 整体监控不产生任何影响,因此,可以灵活扩展 Ganglia 数据收集节点。

2) 收集数据更加精确,不但可以收集实时数据,以图表的形式展示出来,而且允许用户查看历史统计数据,因此,用户可以通过这些数据,做出性能调整、升级、扩容等决策,从而保证应用系统能够满足不断增长的业务需求。

3) 可以通过多播、单播的方式收集数据。在监控的节点较多时通过多播方式收集数据可以大大降低数据收集的负载,提高监控和数据收集性能。而对于不能使用多播收集数据的网络环境,还可以通过单播的方式收集数据,因此 Ganglia 在数据收集方式上非常灵活。

4) 可收集各种度量的数据。Ganglia 默认可收集 CPU、内存、磁盘、IO、处理、网络 6 大方面的数据,同时还提供了 C 或者 Python 接口,用户通过这个接口可以自定义数据收集模块,并且这些模块可以被直接插入 Ganglia 中以监控用户自定义的应用。

基于以上这些优点, Ganglia 非常适合作为监控报警平台的数据收集模块。虽然 Cacti 也可以实现数据的收集和图形报表的展示,但是当监控节点增多时, Cacti 的缺点就慢慢暴露出来,数据收集的准确性、实时性很难得到保障。因此,要构建一个高性能的监控报警平台, Ganglia 是首选的数据收集模块。

9.3 Centreon 作为监控报警模块

有了 Ganglia 收集数据还是不够的,运维人员不可能天天盯着数据报表,因此,还需要对收集到的数据进行监控和报警:对每台需要监控的主机或服务,设置一个报警阈值,当收集到的数据超过这个阈值时,在第一时间能自动报警并通知运维人员,而在收集到的数据没有超过指定的报警阈值时,运维人员就可以去做别的事情,而不用时刻盯着数据报表,这是构建智能监控报警平台必须实现的一个功能。

对主机或服务状态值进行监控,当达到指定阈值时进行报警,要实现这个功能并不是什么难的事情,写个简单的脚本就能实现,但是这样太原始了,没有层次,维护性差,并且当需要监控报警的主机或服务越来越多时,脚本的性能就变得很差,管理也非常不方便,更别说什么可视化效果了,因此,需要有一个专业的监控报警工具来实现这个功能。

Centreon 就是这样一个专业的分布式监控、报警工具，它通过第三方组件可以实现对网络、操作系统和应用程序的监控与报警。在底层，Centreon 通过 nagios 作为监控软件；在数据层，Centreon 通过 ndoutil 模块将监控到的数据定时写入数据库中；在展示层，Centreon 提供了 Web 界面来配置、管理需要监控的主机或服务，并提供多种报警通知方式，同时还可以展现监控数据和报警状态，并且可查询历史报警记录。

关于 Centreon 的介绍和使用，在前面章节已经做过非常详细的介绍，这里不再多说。通过对 Centreon 的使用可知，Centreon 无论在配置、管理、可视化等方面都做得非常专业和完美，并且在多主机、多服务监控的环境下，性能表现也非常稳定，因此，将 Centreon 作为智能监控报警平台的监控报警模块非常适合。

9.4 Ganglia 与 Centreon 的无缝整合

通过前面的介绍，确定了以 Ganglia 作为数据收集模块，Centreon 作为监控报警模块的方案，这样，一个智能监控报警平台两大主要功能模块已经基本实现了。但现在的问题是，如何将收集到的数据传送给监控报警模块呢？这就是数据提取模块要完成的功能。

数据提取模块要完成的功能是：从数据收集模块中定时采集指定的数据，然后将采集到的数据与指定的报警阈值进行比较，如果发现采集到的数据大于或小于指定的报警阈值，那么就通过监控报警模块设置的报警方式进行故障通知。在这个过程中，只有采集数据在数据收集模块中完成，其他操作，例如：采集数据时间间隔、报警阈值设置、报警方式设置、报警联系人设置等都在监控报警模块中完成。

从数据提取模块完成的功能可以看出，此模块主要用来衔接数据收集模块和监控报警模块，进而实现 Ganglia 和 Centreon 的无缝整合。要实现数据提取模块的功能，方法有很多，最简单、最直接的方法就是编写监控脚本，这里提供几个常用的数据提取脚本，然后将脚本添加到 Centreon 中，下面介绍具体的操作过程。

9.4.1 数据提取脚本

这里提供两个数据提取脚本，一个是基于 Python 编写的，一个是基于 PHP 编写的，下面分别进行介绍。

(1) 基于 Python 编写的脚本

这个脚本的原理是通过 Ganglia 提供的数据汇总端口来获取数据，然后将获取到的数据与指定的阈值进行对比，以判断服务是否异常。脚本内容如下：

```
#!/usr/bin/env python
import sys
```

```

import getopt
import socket
import xml.parsers.expat

class GParser:
    def __init__(self, host, metric):
        self.inhost = 0
        self.inmetric = 0
        self.value = None
        self.host = host
        self.metric = metric

    def parse(self, file):
        p = xml.parsers.expat.ParserCreate()
        p.StartElementHandler = parser.start_element
        p.EndElementHandler = parser.end_element
        p.ParseFile(file)
        if self.value == None:
            raise Exception('Host/value not found')
        return float(self.value)

    def start_element(self, name, attrs):
        if name == "HOST":
            if attrs["NAME"]==self.host:
                self.inhost=1
            elif self.inhost==1 and name == "METRIC" and attrs["NAME"]==self.metric:
                self.value=attrs["VAL"]

    def end_element(self, name):
        if name == "HOST" and self.inhost==1:
            self.inhost=0

    def usage():
        print """Usage: check_ganglia_metric \
-h|--host= -m|--metric= -w|--warning= \
-c|--critical= """
        sys.exit(3)

if __name__ == "__main__":
    ganglia_host = '127.0.0.1'
    ganglia_port = 8651
    host = None
    metric = None
    warning = None
    critical = None

```

```

try:
    options, args = getopt.getopt(sys.argv[1:],
        "h:m:w:c:s:p:",
        ["host=", "metric=", "warning=", "critical=", "server=", "port="],
    )
except getopt.GetoptError, err:
    print "check_gmond:", str(err)
    usage()
    sys.exit(3)

for o, a in options:
    if o in ("-h", "--host"):
        host = a
    elif o in ("-m", "--metric"):
        metric = a
    elif o in ("-w", "--warning"):
        warning = float(a)
    elif o in ("-c", "--critical"):
        critical = float(a)
    elif o in ("-p", "--port"):
        ganglia_port = int(a)
    elif o in ("-s", "--server"):
        ganglia_host = a

if critical == None or warning == None or metric == None or host == None:
    usage()
    sys.exit(3)

try:
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((ganglia_host, ganglia_port))
    parser = GParser(host, metric)
    value = parser.parse(s.makefile("r"))
    s.close()
except Exception, err:
    print "CHECKGANGLIA UNKNOWN: Error while getting value \"%s\" % (err)
    sys.exit(3)

if critical > warning:
    if value >= critical:
        print "CHECKGANGLIA CRITICAL: %s is %.2f" % (metric, value)
        sys.exit(2)
    elif value >= warning:
        print "CHECKGANGLIA WARNING: %s is %.2f" % (metric, value)
        sys.exit(1)
    else:

```



```

print "CHECKGANGLIA OK: %s is %.2f" % (metric, value)
sys.exit(0)
else:
    if critical >= value:
        print "CHECKGANGLIA CRITICAL: %s is %.2f" % (metric, value)
        sys.exit(2)
    elif warning >= value:
        print "CHECKGANGLIA WARNING: %s is %.2f" % (metric, value)
        sys.exit(1)
    else:
        print "CHECKGANGLIA OK: %s is %.2f" % (metric, value)
        sys.exit(0)

```

在这个脚本中，需要修改的地方有两个，分别是 `ganglia_host` 和 `ganglia_port`。`ganglia_host` 表示 `gmetad` 服务所在服务器的 IP 地址，Ganglia 可以和 Centreon 安装到一起，也可以分开部署，当 Ganglia 和 Centreon 安装在一起时，这个值就是“127.0.0.1”。`ganglia_port` 表示 `gmetad` 收集数据汇总的交互端口，默认是 8651。

将此脚本命名为 `check_ganglia_metric.py`，然后放到 Centreon 存放 nagios 插件的目录下，默认为 `/usr/lib64/nagios/plugins`，并授予可执行权限。接着介绍此脚本的用法。

在命令行直接执行 `check_ganglia_metric.py` 脚本，即可获得使用帮助：

```

[root@centreonserver plugins]# python check_ganglia_metric.py
Usage: check_ganglia_metric -h|--host= -m|--metric= -w|--warning= -c|--critical=

```

下面分别介绍其中各个参数的意义。

❑ `-h`：表示从哪台主机上提取数据，后跟主机名或 IP 地址。这里需要注意的是，Ganglia 默认将收集的数据存放在 `gmetad` 配置文件中“`rrd_rootdir`”参数指定的目录下，如果被监控的主机没有主机名或主机名没有进行 DNS 解析，那么 Ganglia 就会用此主机的 IP 地址作为目录名来存储收集的数据，反之，就会以主机名作为存储数据的目录名称。因此，这里的“`-h`”参数就要以 Ganglia 存储 `rrd` 数据的目录名称为准。下面是 Ganglia 收集到的 `rrd` 数据的存储结构：

```

[root@centreonserver bestjob]# pwd
/data/rrdsdata/rrds/bestjob
[root@centreonserver bestjob]# ls
192.168.11.188      host0080.bestjob.com  host0078.bestjob.com
192.168.16.10      host0022.bestjob.com  host0065.bestjob.com
[root@centreonserver bestjob]# cd 192.168.11.188
[root@centreonserver 192.168.11.188]# ls
cpu_num.rrd      cpu_system.rrd      disk_free.rrd
load_five.rrd    mem_cached.rrd      mem_total.rrd
cpu_speed.rrd    cpu_user.rrd        disk_total.rrd

```

load_one.rrd mem_free.rrd part_max_used.rrd

- -m: 表示要收集的指标值, 例如 cpu_num、disk_free、cpu_user、disk_total、load_one、part_max_used 等, 这些指标值可以在 Ganglia 存储 rrd 数据的目录中找到, 也可以在 Ganglia 的 Web 界面中查找到。
- -w: 表示警告的阈值, 当此脚本收集的指标值低于或者高于指定的警告阈值时, 此脚本就会发出报警通知, 同时此脚本返回状态值 1。
- -c: 表示故障阈值, 当此脚本收集到的指标值低于或者高于指定的故障阈值时, 此脚本就会发出故障通知, 同时此脚本返回状态值 2。

下面演示一下此脚本的用法, 这里以检测 host0080.bestjob.com 主机的磁盘剩余空间为例, 其他指标值以此类推:

```
[root@centreonserver plugins]# ./check_ganglia_metric.py \
> -h host0080.bestjob.com -m disk_free -w 1000 -c 500
CHECKGANGLIA OK: disk_free is 3045.75
[root@centreonserver plugins]# echo $?
0
[root@centreonserver plugins]# ./check_ganglia_metric.py \
> -h host0080.bestjob.com -m disk_free -w 3045 -c 3000
CHECKGANGLIA WARNING: disk_free is 3043.68
[root@centreonserver nagios]# echo $?
1
[root@centreonserver plugins]# ./check_ganglia_metric.py \
> -h host0080.bestjob.com -m disk_free -w 3050 -c 3045
CHECKGANGLIA CRITICAL: disk_free is 3044.55
[root@centreonserver plugins]# echo $?
2
[root@centreonserver plugins]# ./check_ganglia_metric.py \
> -h host0080.bestjob.com -m part_max_used -w 90 -c 95
CHECKGANGLIA OK: part_max_used is 81.70
```

前三个例子主要用来检测磁盘的剩余空间。在检测磁盘剩余空间时使用的是“disk_free”这个指标, 这个“disk_free”是 host0080.bestjob.com 主机所有磁盘剩余空间的总和, 指标单位为 GB。脚本的执行过程为: 如果剩余磁盘空间正常, 将输出 OK 字样, 同时输出剩余磁盘空间量, 并返回状态值 0; 如果检测磁盘处于 WARNING 状态, 将返回状态值 1; 如果检测磁盘处于 CRITICAL 状态, 将返回状态值 2。这其实就是 nagios 下状态检测脚本的基本写法, nagios 就是通过脚本的返回状态值来判断服务处于何种状态。

接着看最后一个例子, 这个例子用到了“part_max_used”这个指标。这个指标表示磁盘分区的最大使用率, 单位是百分比, 它用来输出系统中磁盘分区使用率的最大值。这个指标非常有用, 经常用来判断系统中某个磁盘分区是否已满。在这个例子中, 指定

WARNING 状态的阈值是 90%，CRITICAL 状态的阈值是 95%，也就是当系统磁盘最大使用率达到 95% 时将发出 CRITICAL 报警。

最后，登录 Ganglia 的 Web 界面，查看 host0080.bestjob.com 主机的“disk_free”和“part_max_used”状态图，检查一下通过 Python 脚本输出的值是否和 Ganglia 生成的状态图一致，如图 9-3 所示。

从图 9-3 可以看出，通过 Python 提取到的数值和 Ganglia 生成的状态图基本一致，如果要查看更详细的统计，可以单击上图进入详细统计页面。在这个详细页面中，可以查看每小时、每天、每周的磁盘状态统计图。另外，通过图 9-3 还可以发现，统计的指标值“disk_free”和“part_max_used”就在状态图的左上角，并配有指标含义解释。

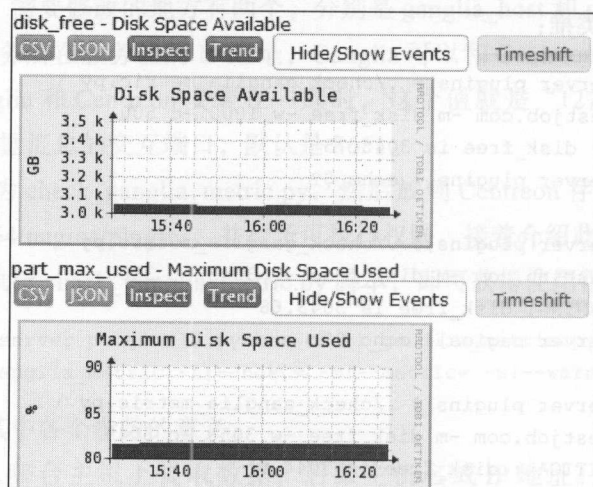


图 9-3 Ganglia 生成的磁盘状态图

(2) 基于 PHP 编写的脚本

这个脚本的原理是通过调用 Ganglia 的 Ganglia Web 页面来获取信息，因此它的功能十分强大，几乎可以获取 Ganglia 收集的任何数据。脚本内容如下：

```
#!/usr/bin/php
<?php
#####
$GANGLIA_WEB="/var/www/html/ganglia";
#####
define("CACHEDATA", 1);
define("CACHEFILE",      "/tmp/nagios/ganglia_metrics");
define("CACHETIME", 45); // How long to cache the data in seconds

if ( $argc !=5 ) {
    echo( "Usage:  $argv[0] <hostname> <metric> <less|more|equal|notequal>
```

```

<critical_value> ie.
    $argv[0] server1 disk_free less 10
    less, more and equal specify whether we mark metric critical if it is less,
    more or equal than critical value
Exiting ....\n");
    exit(3);
}

# To turn on debug set to 1
$debug = 0;
$host = $argv[1];
$metric_name = $argv[2];
$operator = $argv[3];
$critical_value = $argv[4];
global $metrics;

if(CACHEDATA == 1 && file_exists(CACHEFILE)){
    // check for the cached file
    // snag it and return it if it is still fresh
    $time_diff = time() - filemtime(CACHEFILE);
    $expires_in = CACHETIME - $time_diff;
    if( $time_diff < CACHETIME){
        if ( $debug == 1 ) {
            echo("DEBUG: Fetching data from cache. Expires in " . $expires_in . " seconds.\n");
        }
        $metrics = unserialize(file_get_contents(CACHEFILE));
    }
}

if ( ! is_array( $metrics ) ) {

    if ( $debug == 1 ) {
        echo("DEBUG: Querying GMond for new data\n");
    }

    include_once "$GANGLIA_WEB/conf.php";
    # Set up for cluster summary
    $context = "cluster";
    include_once "$GANGLIA_WEB/functions.php";
    include_once "$GANGLIA_WEB/ganglia.php";
    include_once "$GANGLIA_WEB/get_ganglia.php";
    # Put the serialized metrics into a file
    file_put_contents(CACHEFILE, serialize($metrics));
}

# Get a list of all hosts
$ganglia_hosts_array = array_keys($metrics);

```



```

$host_found = 0;

# Find a FQDN of a supplied server name.
for ( $i = 0 ; $i < sizeof($ganglia_hosts_array) ; $i++ ) {
    if ( strpos( $ganglia_hosts_array[$i], $host ) !== false ) {
        $fqdn = $ganglia_hosts_array[$i];
        $host_found = 1;
        break;
    }
}

# Host has been found in the Ganglia tree
if ( $host_found == 1 ) {
    # Check for the existence of a metric
    if ( isset($metrics[$fqdn][$metric_name]['VAL']) ) {
        $metric_value = $metrics[$fqdn][$metric_name]['VAL'];
    } else {
        echo($metric_name . " UNKNOWN - Invalid metric request for this host. Please
        check metric exists.");
        exit(3);
    }

    $ganglia_units = $metrics[$fqdn][$metric_name]['UNITS!'];

    if ( ($operator == "less" && $metric_value > $critical_value) || ( $operator
    == "more" && $metric_value < $critical_value ) || ( $operator == "equal" &&
    trim($metric_value) != trim($critical_value) ) || ( $operator == "notequal" &&
    trim($metric_value) == trim($critical_value) ) ) {
        print $metric_name . " OK - Value = " . $metric_value . " " . $ganglia_units;
        exit (0);
    } else {
        print $metric_name . " CRITICAL - Value = " . $metric_value . " " . $ganglia_units;
        exit (2);
    }
} else {
    echo($metric_name . " UNKNOWN - Hostname info not available. Likely invalid hostname");
    exit(3);
}
?>

```

在这个脚本中，需要修改的是“\$GANGLIA_WEB”变量的值，此变量用来指定 Ganglia Web 的程序路径，这里是 /var/www/html/ganglia，可根据环境进行修改。另外，此脚本调用了 Ganglia Web 程序中的 functions.php、ganglia.php、get_ganglia.php、conf.php

四个 PHP 文件，要确保这四个文件的路径正确。

将此脚本命名为 `check_ganglia_metric.php`，然后放到 Centreon 中存放 nagios 插件的目录下，默认为 `/usr/lib64/nagios/plugins`，并授予可执行权限。接下来介绍此脚本的用法。

在命令行直接执行 `check_ganglia_metric.php` 脚本，即可获得使用帮助：

```
[root@centreonserver plugins]# ./check_ganglia_metric.php
Usage: check_ganglia_metric.php <hostname> <metric> <less|more|equal|notequal> <critical_value>
```

下面分别介绍其中各个参数的含义。

- `hostname`：表示从哪台主机上提取数据，与 `check_ganglia_metric.py` 脚本的“-h”参数含义相同，这里不再解释。
- `metric`：表示要收集的指标值，与 `check_ganglia_metric.py` 脚本的“-m”参数含义相同，也不再说明。
- `less|more|equal|notequal`：表示判断的标准，分别表示小于、大于、等于、不等于，是与指定阈值对比的一个条件。
- `critical_value`：表示故障阈值，当满足指定的阈值条件时，将进行 CRITICAL 状态的报警。

下面演示一下此脚本的用法，这里以检测 `host0080.bestjob.com` 主机的磁盘、内存信息为例，操作过程如下：

```
[root@centreonserver plugins]# ./check_ganglia_metric.php \
> host0080.bestjob.com disk_free less 10
disk_free OK - Value = 2610.908 GB
[root@centreonserver plugins]# ./check_ganglia_metric.php \
> host0080.bestjob.com part_max_used more 95
part_max_used OK - Value = 84.6 %
[root@centreonserver plugins]# ./check_ganglia_metric.php \
> host0080.bestjob.com mem_free less 100000
mem_free OK - Value = 668912 KB
```

首先看第一个例子，利用 `disk_free` 这个指标检测主机 `host0080.bestjob.com` 主机剩余的磁盘空间，其中的告警阈值为 10，单位是 GB，表示磁盘剩余空间低于 10GB 时将发出报警通知。从输出结果发现此主机的磁盘剩余空间还有 2610.908 GB，因此检测结果为 OK 状态。从这个例子可以看出，这个 PHP 脚本在输出结果的同时还输出了结果对应的单位，这个功能非常有用。

其次看第二个例子，通过 `part_max_used` 这个指标检测主机 `host0080.bestjob.com` 磁盘分区的最大使用率，其中告警阈值 95 是个百分比，意思是当磁盘占用率超过 95% 时将发出报警。而通过检测结果发现此主机最大磁盘占用率为 84.6%，因此，检测结果为 OK 状态。

最后看第三个例子，通过 `mem_free` 指标检测 `host0080.bestjob.com` 空闲的物理内存量，从输出结果可知，此主机剩余内存高于指定的告警阈值。

9.4.2 实现 Ganglia 与 Centreon 完美整合

上面介绍了两个常用的数据抽取脚本及其用法，这两个脚本各有优缺点，在实际应用中可以配合使用。要实现对主机的监控报警，最简单的方法是将上面例子中的操作命令写到一个脚本中，然后将这个脚本放到系统守护进程中，定期执行脚本检测即可，但是这种方法不够灵活，无法设定详细的联系人和联系人组，并且维护也不方便。

由于 Centreon 底层调用的是 nagios，因此将这些脚本作为 nagios 的插件，即可实现灵活的报警设置和便捷的管理。接下来以 `check_ganglia_metric.php` 脚本为例，演示如何将此脚本集成到 Centreon 平台中，将其他脚本集成到 Centreon 的方法与此完全相同。

Ganglia 和 Centreon 可以分开独立部署在两台服务器上，也可以部署在一台服务器上，这里设定 Ganglia 和 Centreon 部署在一台服务器上。结合前面介绍的 Centreon 知识，在监控服务器的 Centreon Web 界面，选择“Configuration → Commands → Checks”，然后单击“Add”新建一个 Command，如图 9-4 所示。

图 9-4 创建一个名为 `check_ganglia_php` 的 Command

在图 9-4 中，创建了一个名为 `check_ganglia_php` 的 Command，选择“Command Type”为“Check”。接着重点看“Command Line”的内容，其中，“`$USER1$`”就是 Centreon 服务器上 nagios 监控插件的路径，默认是“`/usr/lib64/nagios/plugins`”，前面已经把这个监控脚本放到了此路径下，这里直接引用即可。在这个脚本对应的参数中定义了三个参数变量“`$ARG1$`”、“`$ARG2$`”和“`$ARG3$`”，分别用于指定 Ganglia metric、警告条件和 CRITICAL 阈值。这里建议将每个参数变量的作用都在“Argument Descriptions”选项上做

个描述，以保证在添加服务时不容易出错。

这样，就将 `check_ganglia_metric.php` 脚本作为一个命令集成到 Centreon 中了。接下来演示如何通过这个脚本检测一批主机的磁盘空间最大占用率。首先在 Centreon Web 界面选择“Configuration → Services → Services by host group”，单击“Add”添加一个主机组服务，如图 9-5 所示。

Service Configuration		Relations	Data Processing	Service Extended Info								
Add a Service												
General Information												
Description *	check_disk_group											
Service Template	generic-service											
Service State												
Is Volatile	<input type="radio"/> Yes <input checked="" type="radio"/> No <input type="radio"/> Default											
Check Period *	24x7											
Check Command *	check_ganglia_php											
Args	<table border="1"> <thead> <tr> <th>Argument</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>ARG1</td> <td>part_max_used</td> </tr> <tr> <td>ARG2</td> <td>more</td> </tr> <tr> <td>ARG3</td> <td>95</td> </tr> </tbody> </table>				Argument	Value	ARG1	part_max_used	ARG2	more	ARG3	95
Argument	Value											
ARG1	part_max_used											
ARG2	more											
ARG3	95											
Max Check Attempts *												
Normal Check Interval *	30 * 60 seconds											
Retry Check Interval *												
Active Checks Enabled	<input type="radio"/> Yes <input checked="" type="radio"/> No <input type="radio"/> Default											
Passive Checks Enabled	<input type="radio"/> Yes <input checked="" type="radio"/> No <input type="radio"/> Default											

图 9-5 添加一个名为 `check_disk_group` 的主机组服务

这里先看一下“Service Configuration”标签中的几个选项：“Description”就是监控服务的名称，这里定义为“`check_disk_group`”；“Service Template”是指定服务的模板，这里仍然选择“`generic-service`”；“Check Command”选项用来指定服务检查的命令，从下拉菜单中选择刚才创建的命令“`check_ganglia_php`”即可；最后一个选项“Args”就是刚才创建 `check_ganglia_php` 命令时指定的三个变量，根据每个变量的含义，依次填写 Ganglia metric、警告条件和 CRITICAL 阈值。

接下来看报警通知的设置，如图 9-6 所示。

在图 9-6 中，“Notification Enabled”选项表示是否启用报警通知，这里选择“Yes”。“Implied Contacts”选项表示指定报警联系人，如果某些服务仅需要通知很少的人员，可以在这里指定对应的报警联系人。如果报警联系人很多，一个一个添加就显得非常麻烦，后续的增加、删除报警联系人都非常繁琐，此时可以使用“Implied Contact Groups”选项，将不同类型的联系人分组，然后指定需要接收报警信息的联系人组即可，这样整个联系人组下的所有人员都能收到报警邮件了。

The screenshot shows the 'Notification' configuration window for the 'check_disk_group' service. It includes the following settings:

- Notification Enabled:** Radio buttons for Yes (selected), No, and Default.
- Implied Contacts:** A list of 'Available' contacts (Guest, Supervisor, User) and a 'Selected' list containing 'ixdha'. 'Add' and 'Remove' buttons are present.
- Implied Contact Groups:** A list of 'Available' groups (Guest) and a 'Selected' list containing 'Supervisors' and 'View_group'. 'Add' and 'Remove' buttons are present.
- Notification Interval:** A text input field with '30' and a label '* 60 seconds'.
- Notification Period:** A dropdown menu showing '24x7'.
- Notification Type:** Checkboxes for Warning (checked), Unknown (checked), Critical (checked), Recovery (checked), Flapping (unchecked), and Downtime Scheduled (unchecked).
- First notification delay:** A text input field with '0' and a label '* 60 seconds'.

图 9-6 check_disk_group 服务中报警通知的设置

此外，“Notification Interval”选项用来设置发送报警通知的时间间隔，这个值应该大于或等于图 9-5 中为“Normal Check Interval”选择的值。“Notification Period”表示报警周期，建议选择 7×24 的生成环境。“Notification Type”表示发送什么类型的报警通知，经常选择的有 Warning、Critical、Recovery 等，可以根据具体环境来选择。最后一个选项“First notification delay”表示发生故障后推迟多久发送报警通知，如果输入 0，那么表示故障发生后立刻发送报警通知，可根据情况进行设置。

下面继续看“Relations”标签，如图 9-7 所示。

The screenshot shows the 'Relations' configuration window under the 'Service Configuration' tab. It includes the following settings:

- Linked with Host Groups:** A list of 'Available' host groups (Centos-server-group, Centos1-server-group, Firewall, Linux-Servers, Networks, Printers, Routers) and a 'Selected' list containing 'bestjob.com'. 'Add' and 'Remove' buttons are present.
- Parent Service Groups:** A list of 'Available' parent service groups and a 'Selected' list. 'Add' and 'Remove' buttons are present.

At the top right, there are 'Save' and 'Reset' buttons.

图 9-7 指定需要监控的主机组

如果需要监控很多主机的同一个服务时，向主机一个一个添加服务就变得十分烦琐，此时通过添加主机组的方式，一次性完成主机组下所有主机的监控，简单而灵活。

“Relations” 标签可以用来指定要监控的主机组，在图 9-7 左边的主机组列表中选择要监控的主机组即可，这里选择 bestjob.com 主机组。在完成所有设置后，单击“Save”保存退出，check_disk_group 服务添加完成。

至此，已经完成 check_ganglia_metric.php 脚本和 Centreon 的集成，其实也就是 Ganglia 和 Centreon 的集成，而这个脚本就是两者集成的桥梁而已。

在完成所有配置后，需要重启 Centreon 服务，这样所有的配置和修改才能生效。在重启 Centreon 服务后，查看 check_disk_group 服务的运行状态，如图 9-8 所示。

从图 9-8 中可以看出，bestjob.com 主机组中包含了四台主机，每台主机的磁盘最大占用率都处于正常状态，并且输出了目前磁盘最大占用率的比值。由此可知，check_ganglia_metric.php 脚本集成到 Centreon 后工作正常，完美实现了从 Ganglia 采集数据，在 Centreon 中设置报警规则的监控、报警一体化过程。

Hosts	Services	Status	Duration	Last Check	Tries	Status information
Centreon-Server	check_disk_group	OK	6m 45s	10/02/2014 17:42:45	1/3 (H)	part_max_used OK - Value = 29.2 %
cloud1	check_disk_group	OK	6m 7s	10/02/2014 17:43:23	1/3 (H)	part_max_used OK - Value = 24.5 %
node1	check_disk_group	OK	5m 6s	10/02/2014 17:44:24	1/3 (H)	part_max_used OK - Value = 69.5 %
node3	check_disk_group	OK	5m 55s	10/02/2014 17:43:35	1/3 (H)	part_max_used OK - Value = 24.5 %

图 9-8 check_disk_group 服务运行状态

9.5 在 Centreon 中实现批量数据收集与监控报警

上一节主要介绍了如何通过数据抽取脚本从 Ganglia 中抽取数据，然后通过脚本的判断最终实现在 Centreon 上的报警通知。这个过程看似很完美，其实隐藏着一些问题，比如，从给出的两个数据抽取文件中，脚本每次只能检测一台主机的一个服务状态，如果要检测多台主机的多个服务状态，就需要多次重复执行这个脚本。例如，要检测 100 台主机的磁盘最大占用率，脚本就要重复执行 100 次，同理，要检测 100 台主机中的 10 个服务状态，脚本就要执行 1000 次，在 Centreon 平台中，脚本检测是定期执行的，如果每小时执行一次检测，那么每小时就要执行脚本 1000 次。由此可知，这种脚本执行方式效率低下，严重浪费服务器资源，在监控主机较少的环境中，监控效率还勉强能够接受，但是当监控的主机超过 500 台或更多，监控的服务超过 100 或更多时，监控效率将更加低下，监控脚本从 Ganglia 抽取数据的时间也将变得很长，可能还会发生获取监控数据超时的情况，对于要求

监控精度很高、报警及时性强的监控报警平台来说,这是不可容忍的。

如果能减少监控脚本重复执行检查的次数,或者让脚本一次检查多台服务器的多个服务状态,那么脚本的执行效率将大大提升。在上节介绍的第二个脚本中,是通过读取 Ganglia Web 的页面信息来获取监控数据的,既然通过此脚本能一次获取某台主机的信息,那么也能一次获取多台主机的数据。非常幸运, Ganglia 本身已经实现了这个功能,这里只需稍作修改拿过来使用即可。在 Ganglia Web 的程序目录下有一个 nagios 目录,其中有多 shell 脚本和 PHP 脚本,这里重点介绍下 check_host_regex.sh 和 check_host_regex.php 这两个脚本,其他脚本的使用方法与此类似。

经过修改后的 check_host_regex.sh 脚本内容如下:

```

GANGLIA_URL="http://localhost/ganglia/nagios/check_host_regex.php"

# Build the rest of the arguments into the arg string for the URL.
CHECK_ARGS=''
if [ "$#" -gt "0" ]
then
    CHECK_ARGS=$1
    shift
    for ARG in "$@"
    do
        CHECK_ARGS=${CHECK_ARGS}"&"${ARG}
    done
else
    echo "Sample invocation $0 hreg=web|apache checks=load_one,more,1:load_five,
more,2 ignore_unknowns=0"
    echo "    Set ignore_unknowns=1 if you want to ignore hosts that don't posses
a particular metric."
    echo "    This is useful if you want to use a catchall regex e.g. .* however
some hosts lack a metric"
    exit 1
fi

RESULT=`curl -s "${GANGLIA_URL}?${CHECK_ARGS}"`
EXIT_CODE=`echo $RESULT | cut -f1 -d'|'`

REST=`echo $RESULT | cut -f2 -d'|'`
for x in $EXIT_CODE; do
    case $x in
        OK)
            echo $REST
            exit 0;;
        WARNING)

```

图 9-7 为已经安装好的主机图

```

echo $REST
exit 1;;
CRITICAL)
echo $REST
exit 2;;
*)
echo $REST
exit 1;;
esac
done

```

这里主要看下此文件第一行“`GANGLIA_URL`”的内容，这个变量用于指定 HTTP 方式访问 `check_host_regex.php` 脚本的路径，后面跟的 URL 只要服务器本身能够访问即可。从脚本内容看，`check_host_regex.sh` 主要用于对获取的监控数据进行判断，而 `check_host_regex.php` 脚本主要用来获取监控数据。`check_host_regex.php` 脚本获取数据的方式与上节介绍的 `check_ganglia_metric.php` 脚本实现原理完全相同，实现起来非常简单，这里不再讲述。

下面介绍 `check_host_regex.sh` 脚本的用法。

直接在命令行执行 `check_host_regex.sh` 脚本，即可显示详细用法，例如：

```

[root@centreonserver plugins]# ./check_host_regex.sh
Sample invocation ./check_host_regex.sh hreg=Hostname checks=load_one,more,1:
load_five,more,2 ignore_unknowns=0
Set ignore_unknowns=1 if you want to ignore hosts that don't posses a
particular metric.

```

下面介绍其中各个参数的意义。

- **hreg**：后面跟主机名或主机名标识。这个参数的含义与 `check_ganglia_metric.php` 脚本中“`hostname`”参数的含义相同，但用法稍有不同。如果在 `hreg` 后面指定一个完整的主机名，那么就收集这台主机的状态信息。同时，`hreg` 参数还支持正则表达式，只需提供一个主机名的标识，就可以批量检测包含此标识的所有主机，例如，有 800 台主机，主机名都包含有“`bestjob`”这个字符，那么只需设置“`hreg=bestjob`”即可实现一个脚本一次检查 800 台主机的服务状态。
- **checks**：后面跟检测服务的指标值、检查条件和告警阈值。常见的服务指标有 `load_one`、`disk_free`、`swap_free`、`part_max_used` 等，检查条件有 `more`（大于）、`less`（小于）、`equal`（等于）、`notequal`（不等于）四种，告警阈值根据实际应用环境来设置。这个参数还有个功能就是可以一次设置多个服务状态，每个服务之间用“`:`”进行分割即可。有了这个功能，就可以通过一个脚本一次检测多个服务的运行状态，大大提高了脚本检测的效率。

□ `ignore_unknowns` : 此参数用来设置是否忽略 UNKNOWN 状态的服务。设置此参数为 1 表示忽略 UNKNOWN 状态的服务, 反之, 设置为 0 表示不忽略。

下面演示一下此脚本的用法, 这里以检测主机名含有 “bestjob” 标识的主机为例, 操作过程如下:

```
[root@centreonserver plugins]# ./check_host_regex.sh \
> hreg=bestjob checks=load_one,more,15
Services OK = 796, CRIT/UNK = 4 ;
CRITICAL host0089.bestjob.com load_one = 16.96,
CRITICAL host0133.bestjob.com load_one = 22.91,
CRITICAL host0028.bestjob.com load_one = 15.02,
CRITICAL host0329.bestjob.com load_one = 16.68
```

此命令用来检测主机名含有 “bestjob” 标识的主机 1 分钟内的负载状态, 当负载超过 15 时进行告警通知。从输出可知, 主机名含有 “bestjob” 标识的主机共有 800 台, 其中 4 台主机在 1 分钟内的负载状态超过 15, 并给出了超载主机的主机名和当前的负载状态值。

下面是一个脚本检测多台主机的多个服务的用法, 操作过程如下:

```
[root@centreonserver plugins]# ./check_host_regex.sh \
> hreg=bestjob checks=load_one,more,15:disk_free,less,900 ignore_unknowns=1
Services OK = 787, CRIT/UNK = 13 ;
CRITICAL host0081.bestjob.com load_one = 25.51 ,
CRITICAL host0246.bestjob.com load_one = 16.86 ,
CRITICAL host0003.bestjob.com disk_free = 576.318 GB,
CRITICAL dbmysql.bestjob.com disk_free = 520.721 GB,
CRITICAL webapp.bestjob.com disk_free = 461.966 GB,
CRITICAL host0200.bestjob.com disk_free = 852.420 GB,
CRITICAL host0055.bestjob.com disk_free = 279.465 GB,
CRITICAL dbdata.bestjob.com disk_free = 636.190 GB,
CRITICAL webui.bestjob.com disk_free = 525.538 GB,
CRITICAL server0232.bestjob.com disk_free = 861.330 GB,
CRITICAL server0159.bestjob.com disk_free = 801.443 GB,
CRITICAL host0080.bestjob.com disk_free = 739.467 GB,
CRITICAL etlserver.bestjob.com disk_free = 826.477 GB
```

此命令检测含有 “bestjob” 标识在主机 1 分钟内的负载状态和剩余空闲磁盘空间情况, 并忽略 UNKNOWN 状态的服务。从检测结果中可以发现, 在 800 台主机中, 有 13 台主机出现负载或磁盘空闲空间告警问题, 并输出了详细的告警信息。

最后, 还需要将此脚本集成到 Centreon 平台上, 以实现主机和服务的批量监控与批量报警。集成方法与上节介绍的集成 `check_ganglia_metric.php` 的方法完全相同, 这里不过多介绍。

在完成脚本集成后,重启 Centreon 服务,即可实现主机和服务的批量监控与报警,通过批量的方式监控主机状态。对于 500 台以上主机的运维环境来说,如果需要监控 100 个服务,那么每个脚本监控 20 个服务,脚本执行 5 次即可完成所有主机服务状态的监控,大大减少了脚本的执行次数,同时每个脚本的执行时间也不会显著增加。实践证明,通过批量监控的方式基本解决了大运维环境下的监控报警性能问题。

图 9-9 是 Centreon 在批量监控下的一个运行状态截图,通过此图可以清晰地看出哪些主机出现了告警问题,以及服务上次检测的时间和下次检测的时间。

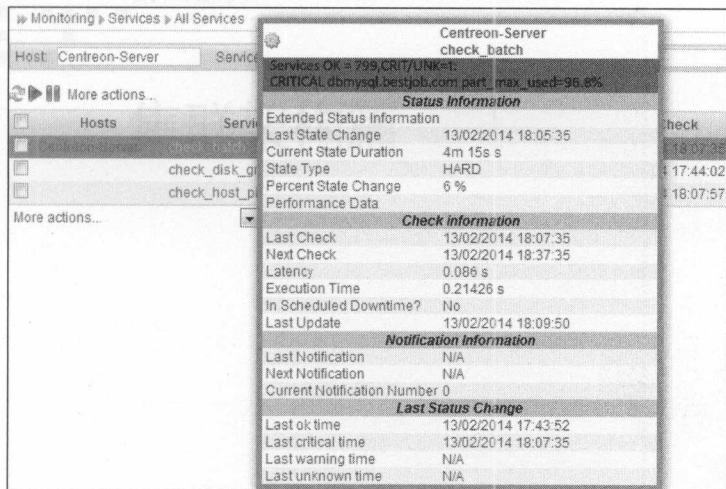


图 9-9 Centreon 在批量监控服务下的运行状态图

由图 9-9 可知,此服务批量监控了 800 台主机的“part_max_used”指标,其中,799 台主机的“part_max_used”状态正常,主机 dbmysql.bestjob.com 的磁盘最大占用率超过了指定的告警阈值。

如果对服务故障设置了发送邮件的报警通知,那么此时就会收到邮件报警通知,收到的邮件内容类似如下:

```
*****centreon Notification *****
```

```
Notification Type: PROBLEM
```

```
Service: part_max_used
```

```
Host: Centreon-Server
```

```
Address: 192.168.20.98
```

```
State: CRITICAL
```

```
Date/Time: Thu Feb 13 18:20:54 CST 2014
```

```
Additional Info:
```

```

Services OK = 799, CRIT/UNK = 1:
CRITICAL dbmysql.bestjob.com part_max_used=96.8%

```

如果要修改邮件报警的内容,就需要在 Centreon 中修改 service-notify-by-email 命令,具体修改方式在 8.5.5 节中已经做过详细介绍,这里不再说明。

至此,在 Centreon 中实现批量数据收集和监控报警的方法已经介绍完毕。总体来说,都是借助现有的脚本稍作修改实现的,整个过程比较简单。

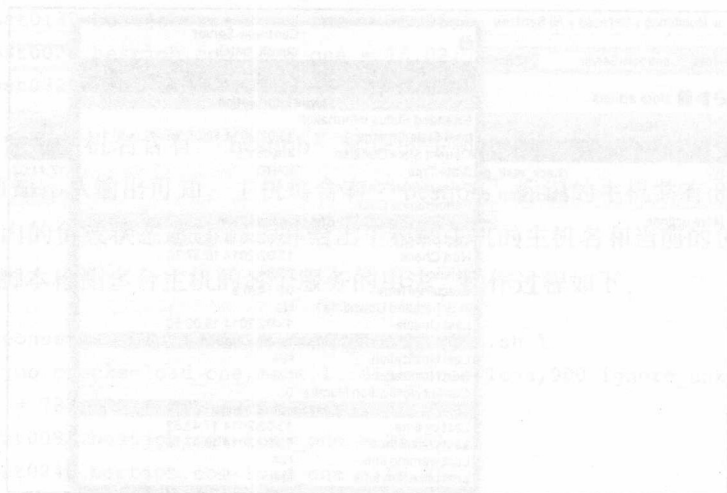


图 9-9 Centreon 中批量数据收集和监控报警

由图 9-9 可知,此脚本批量收集了 800 台主机的“part_max_used”状态,其中 799 台主机的“part_max_used”状态正常,主机 dbmysql.bestjob.com 的“part_max_used”状态为“CRITICAL”,即数据库使用率过高,需要报警。

如果对脚本做如下修改,即可实现报警内容的自定义。

```

*****centreon Notification*****
Notification Type: PROBLEM

```

此命令的脚本含有“bestjob”标识,主机 1 分钟内的负载状态有异常,即报警。并忽略 UNKNOWN 状态的服务器。从报警结果中可以发现,在 800 台主机中,有 1 台主机出现负载或磁盘空间占用异常问题,并输出了详细的告警信息。

最后,还需要将此脚本集成到 Centreon 平台上,以实现主机和服务的批量数据收集和报警。集成方法与上节介绍的集成 dbmysql.bestjob.com 类似,这里不再赘述,感兴趣者可以参考。

第四部分 Part 4

集群架构篇

10.1 高性能Web服务器Nginx介绍

Nginx 是一个高性能的 HTTP 和反向代理服务软件，也是一个 IMAP/POP3/SMTP 代理服务器。Nginx（发音是“engine x”）由俄罗斯程序员设计师 Igor Yarov 开发（Igor 将源代码以类 BSD 许可证的形式发布），可以运行在 UNIX、GNU/Linux、BSD、Mac OS X、Solaris，以及 Microsoft Windows 等操作系统中。随着 Nginx 在越来越多的网站被采用，其稳定、高效的特性逐渐被越来越多的用户认可。

10.1.1 Nginx 的组成与工作原理

Nginx 由内核和模块组成。Nginx 的工作非常简单，仅仅通过查找配置文件中一个命令，用于完成相应的工作。

- 第10章 高性能Web服务器Nginx
- 第11章 高性能集群软件Keepalived
- 第12章 千万级高并发负载均衡软件HAProxy
- 第13章 构建高性能的MySQL集群系统
- 第14章 高性能负载均衡集群软件HAProxy

Nginx 的模块从结构上分为基本模块、核心模块和第三方模块。其中，HTTP 模块、EVENT 模块和 MAIL 模块等属于核心模块，HTTP Access 模块、HTTP FastCGI 模块、HTTP Proxy 模块和 HTTP Rewrite 模块属于基本模块，而 HTTP Upstream Request Hash 模块、Notice 模块和 HTTP Access Key 模块属于第三方模块。用户根据自己的需要开发的模块都属于第三方模块。正是有了这些模块的支持，Nginx 的功能才会如此强大。

Nginx 的模块从功能上分为三类，如下所示。

高性能 Web 服务器 Nginx

10.1 高性能 Web 服务器 Nginx 介绍

Nginx 是一个高性能的 HTTP 和反向代理服务器软件，也是一台 IMAP/POP3/SMTP 代理服务器。Nginx（发音是“engine x”）由俄罗斯程序设计师 Igor Sysoev 开发（Igor 将源代码以类 BSD 许可证的形式发布），可以运行在 UNIX、GNU/Linux、BSD、Mac OS X、Solaris，以及 Microsoft Windows 等操作系统中。随着 Nginx 在很多大型网站稳定运行，其稳定、高效的特性逐渐被越来越多的用户认可。

10.1.1 Nginx 的组成与工作原理

Nginx 由内核和模块组成，其中，内核的设计非常微小和简洁，完成的工作也非常简单，仅仅通过查找配置文件将客户端请求映射到一个 location block（location 是 Nginx 配置中的一个命令，用于 URL 匹配），而在这个 location 中所配置的每个命令将会启动不同的模块去完成相应的工作。

Nginx 的模块从结构上分为核心模块、基础模块和第三方模块，其中，HTTP 模块、EVENT 模块和 MAIL 模块等属于核心模块，HTTP Access 模块、HTTP FastCGI 模块、HTTP Proxy 模块和 HTTP Rewrite 模块属于基本模块，而 HTTP Upstream Request Hash 模块、Notice 模块和 HTTP Access Key 模块属于第三方模块，用户根据自己的需要开发的模块都属于第三方模块。正是有了这么多模块的支撑，Nginx 的功能才会如此强大。

Nginx 的模块从功能上分为三类，如下所示。

- ❑ **Handlers (处理器模块)**。此类模块直接处理请求，并进行输出内容和修改 headers 信息等操作。handlers 处理器模块一般只能有一个。
- ❑ **Filters (过滤器模块)**。此类模块主要对其他处理器模块输出的内容进行修改操作，最后由 Nginx 输出。
- ❑ **Proxies (代理类模块)**。就是 Nginx 的 HTTP Upstream 之类的模块，这些模块主要与后端一些服务比如 fastcgi 等操作交互，实现服务代理和负载均衡等功能。

图 10-1 展示了在 Nginx 的模块下一次常规的 HTTP 请求和响应的过程。

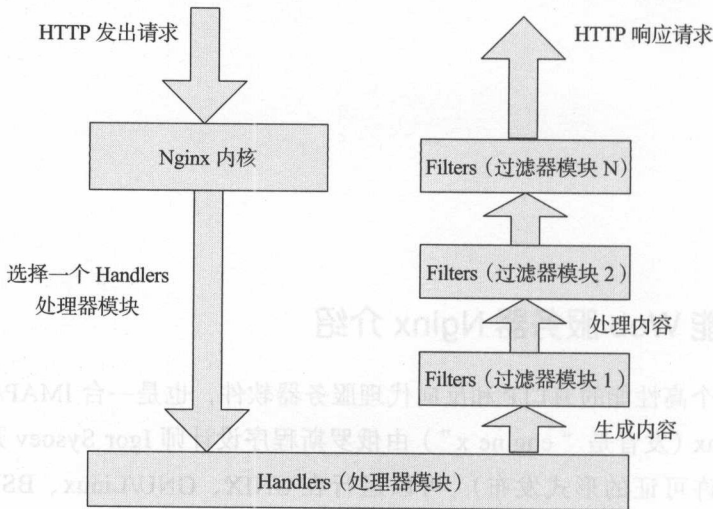


图 10-1 Nginx 的 HTTP 模块请求和响应过程

在工作方式上，Nginx 分为单工作进程和多工作进程两种模式。在单工作进程模式下，除主进程外，还有一个单线程的工作进程；在多工作进程模式下，每个工作进程包含多个线程。Nginx 默认为单工作进程模式。

Nginx 的模块直接被编译进 Nginx，因此属于静态编译方式。在启动 Nginx 后，自动加载 Nginx 的模块，不像在 Apache 中一样，首先将模块编译为一个 so 文件，然后在配置文件中指定是否加载。在解析配置文件时，Nginx 的每个模块都有可能去处理某个请求，但是同一个处理请求只能由一个模块来完成。

10.1.2 Nginx 的性能优势

如今，Nginx 已经非常流行和普及了，几乎所有的大型门户网站都在使用 Nginx。Nginx 作为 HTTP 服务器的优势显而易见，它有很多其他 Web 服务器无法比拟的性能和优势：

- ❑ 作为 Web 服务器，Nginx 处理静态文件、索引文件以及自动索引效率非常高。

- 作为代理服务器, Nginx 可以实现快速、高效的反向代理, 提升网站性能。
- 作为负载均衡服务器, Nginx 既可以在内部直接支持 Rails 和 PHP, 也可以支持 HTTP 代理服务器, 对外进行服务。同时支持简单的容错和利用算法进行负载均衡。
- 在性能方面, Nginx 是专门为性能优化而开发的, 在实现上非常注重效率。它采用内核 Poll 模型, 可以支持更多的并发连接, 并在大并发时占用很低的内存资源。
- 在稳定性方面, Nginx 采取了分阶段资源分配技术, 使得对 CPU 与内存的占用率非常低。
- 在高可用性方面, Nginx 支持热部署, 启动速度迅速, 因此可以在不间断服务的情况下, 对软件版本或者配置进行升级, 即使运行数月也无需重新启动, 几乎可以做到 7×24 小时不间断运行。

10.2 Nginx 的安装

Nginx 的官方网站是 <http://nginx.org/>, 从这里可以获得 Nginx 的最新版本信息。Nginx 有三个版本: 稳定版、开发版和历史稳定版。开发版更新较快, 包含最新功能和 bug 修复, 但同时也可能会遇到新的 bug, 开发版一旦更新稳定下来, 就会被加入稳定版分支中。然而有些新功能不一定会被加到旧的稳定版中。稳定版本更新较慢, 但是 bug 较少, 可以作为生产环境的首选, 因此通常建议使用稳定版。历史稳定版本为以往稳定版本的汇总, 不包含最新的功能。

本章选择当前的稳定版本 nginx-1.4.7 作为介绍对象, 开始介绍编译安装。在安装 Nginx 之前, 确保系统已经安装了 gcc、openssl-devel、pcre 和 zlib-devel 软件库。

Linux 开发库是在安装系统时手动选择安装的, gcc、openssl-devel、zlib-devel 三个软件库可以通过安装光盘直接选择安装得到, 而 pcre 库默认不在系统光盘中, 所以这里重点介绍 pcre 库的安装。

10.2.1 安装 Nginx 依赖库

安装 pcre 库是为了使 Nginx 支持 HTTP Rewrite 模块。安装非常简单, 过程如下:

```
[root@localhost home]# tar zxvf pcre-8.34.tar.gz
[root@localhost home]# cd pcre-8.34
[root@localhost pcre-8.34]# ./configure
[root@localhost pcre-8.34]# make
[root@localhost pcre-8.34]# make install
```

这样, pcre 库就安装完成了。

10.2.2 快速安装 Nginx

Nginx 的安装非常简单。在默认情况下，经过编译安装的 Nginx 包含了大部分可用模块。可以通过“./configure --help”选项设置各个模块的使用情况，例如，对不需要的 http_ssi 模块，可通过“--without-http_ssi_module”方式关闭，同理，如果需要“http_perl”模块，那么可以通过“--with-http_perl_module”方式来安装。安装过程如下：

```
[root@localhost home]# tar zxvf nginx-1.4.7.tar.gz
[root@localhost home]# cd nginx-1.4.7
[root@localhost nginx-1.4.7]# ./configure --prefix=/usr/local/nginx \
> --with-http_stub_status_module --with-http_gzip_static_module
[root@localhost nginx-1.4.7]# make
[root@localhost nginx-1.4.7]# make install
```

在上面的 configure 选项中，“--with-http_stub_status_module”可以用来启用 Nginx 的 NginxStatus 功能，以监控 Nginx 的当前状态。同理，“--with-http_gzip_static_module”表示启用 HttpGzip 模块。

至此，Nginx 安装完成。

10.3 配置与调试 Nginx

10.3.1 Nginx 配置文件结构

Nginx 的配置文件是一个纯文本文件，它一般位于 Nginx 安装目录的 conf 目录下，整个配置文件是以 block 的形式组织的。每个 block 一般以一个大括号“{}”来表示。block 可以分为几个层次，整个配置文件中 Main 命令位于最高层，在 Main 层下面可以有 Events、HTTP 等层级，而在 HTTP 层中又包含 Server 层，即 server block，server block 中又可分为 location 层，并且一个 server block 中可以包含多个 location block。

一个完整的配置文件结构如图 10-2 所示。

在了解配置文件结构之后，就可以开始配置和调试 Nginx 了。

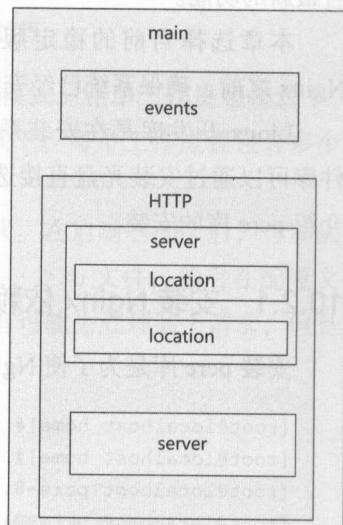


图 10-2 Nginx 配置文件结构

10.3.2 Nginx 配置文件详解

Nginx 安装完毕后，会产生相应的安装目录，根据前面的安装路径，Nginx 的配置文

件路径为 `/usr/local/nginx/conf`，其中 `nginx.conf` 为 Nginx 的主配置文件。这里重点介绍下 `nginx.conf` 这个配置文件。

Nginx 配置文件主要分成四个部分：`main`（全局设置）、`server`（主机设置）、`upstream`（负载均衡服务器设置）和 `location`（URL 匹配特定位置的设置）。`main` 部分设置的命令将影响其他所有设置；`server` 部分的命令主要用于指定主机和端口；`upstream` 命令主要用于负载均衡，设置一系列的后端服务器；`location` 部分用于匹配网页位置。这四者之间的关系式：`server` 继承 `main`，`location` 继承 `server`，`upstream` 既不会继承其他设置也不会被继承。

在这四个部分当中，每个部分都包含若干命令，这些命令主要包含 Nginx 的主模块命令、事件模块命令、HTTP 核心模块命令，同时每个部分还可以使用其他 HTTP 模块命令，例如 `Http SSL` 模块、`HttpGzip Static` 模块和 `Http Addition` 模块等。

下面通过一个 Nginx 配置实例详细介绍 `nginx.conf` 每个命令的含义。一个配置好的 `nginx.conf` 文件内容如下：

```
user nobody nobody;
worker_processes 4;
error_log logs/error.log notice;
pid logs/nginx.pid;
worker_rlimit_nofile 65535;
events{
    use epoll;
    worker_connections 65536;
}
```

这段代码用于对 Nginx 的全局属性进行配置，每个参数含义如下。

- ❑ `user`：主模块命令，指定 Nginx 的 worker 进程运行用户以及用户组，默认由 `nobody` 账号运行。
- ❑ `worker_processes`：指定 Nginx 要开启的进程数。
- ❑ `error_log`：用来定义全局错误日志文件。日志输出级别有 `debug`、`info`、`notice`、`warn`、`error`、`crit` 可供选择，其中，`debug` 输出日志最为详细，而 `crit` 输出日志最少。
- ❑ `pid`：用来指定进程 id 的存储文件位置。
- ❑ `worker_rlimit_nofile`：用于指定一个 nginx 进程可以打开的最多文件描述符数目，这里是 65535，需要使用命令“`ulimit -n 65535`”在系统中进行设置。
- ❑ `events`：设定 Nginx 的工作模式及连接数上限。其中参数“`use`”用来指定 Nginx 的工作模式，Nginx 支持的工作模式有 `select`、`poll`、`kqueue`、`epoll`、`rtsig` 和 `/dev/poll`。其中 `select` 和 `poll` 都是标准的工作模式，`kqueue` 和 `epoll` 是高效的工作模式，对于 Linux 系统，`epoll` 工作模式是首选。而参数“`worker_connections`”用于定义 Nginx 每个进程的最大连接数，默认是 1024。最大客户端连接数由 `worker_processes` 和 `worker_connections` 决定，即 $\text{max_client} = \text{worker_processes} * \text{worker_connections}$ 。

进程的最大连接数受 Linux 系统进程的最大打开文件数限制, 在执行操作系统命令“`ulimit -n 65536`”后 `worker_connections` 的设置才能生效。

下面这段内容是 Nginx 对 HTTP 服务器相关属性的配置:

```
http{
include      conf/mime.types;
default_type application/octet-stream;
log_format  main '$remote_addr - $remote_user [$time_local] '
    '"$request" $status $bytes_sent '
    '"$http_referer" "$http_user_agent" '
    '"$gzip_ratio"';
log_format  download '$remote_addr - $remote_user [$time_local] '
    '"$request" $status $bytes_sent '
    '"$http_referer" "$http_user_agent" '
    '"$http_range" "$sent_http_content_range"';

access_log  logs/www.ixdba.net.access.log  main;

client_max_body_size          20m;
client_header_buffer_size     32k;
large_client_header_buffers   4 128k;
sendfile on;
tcp_nopush                    on;
tcp_nodelay                    on;
keepalive_timeout             60;
client_header_timeout         10;
client_body_timeout           10;
send_timeout                   10;
```

上述代码中每个配置选项的含义如下。

- ❑ `include` : 主模块命令, 实现对配置文件所包含文件的设定, 可以减少主配置文件的复杂度。类似于 Apache 中的 `include` 方法。
- ❑ `default_type` : 属于 HTTP 核心模块命令, 这里设定默认类型为二进制流, 也就是当文件类型未定义时使用这种方式, 例如, 在没有配置 PHP 环境时, Nginx 是不予解析的, 此时, 用浏览器访问 PHP 文件就会出现下载窗口。
- ❑ `log_format` : 用于指定 Nginx 日志的输出格式。`main` 为此日志输出格式的名称, 可以在下面的 `access_log` 命令中引用。
- ❑ `client_max_body_size` : 用来设置允许客户端请求的最大的单个文件字节数。
- ❑ `client_header_buffer_size` : 用于指定来自客户端请求头的 `headerbuffer` 大小。对于大多数请求, 1KB 的缓冲区大小已经足够, 如果自定义了消息头或有更大的 cookie, 可以增加缓冲区大小。这里设置为 32KB。

- ❑ `large_client_header_buffers`：用来指定客户端请求中较大的消息头的缓存最大数量和大小，“4”为个数，“128K”为大小，最大缓存量为 4 个 128KB。
- ❑ `sendfile`：用于开启高效文件传输模式。将 `tcp_nopush` 和 `tcp_nodelay` 两个命令设置为“on”用于防止网络阻塞。
- ❑ `keepalive_timeout`：设置客户端连接保持活动的超时时间。在超过这个时间之后，服务器会关闭该连接。
- ❑ `client_header_timeout`：设置客户端请求头读取超时时间。如果超过这个时间，客户端还没有发送任何数据，Nginx 将返回“Request time out (408)”错误。
- ❑ `client_body_timeout`：设置客户端请求主体读取超时时间。如果超过这个时间，客户端还没有发送任何数据，Nginx 将返回“Request time out (408)”错误，默认值是 60。
- ❑ `send_timeout`：指定响应客户端的超时时间。这个超时仅限于两个连接活动之间的时间，如果超过这个时间，客户端没有任何活动，Nginx 将会关闭连接。

下面这段代码是 HttpGzip 模块在 Nginx 配置中的相关属性设置：

```
gzip on;
gzip_min_length 1k;
gzip_buffers 4 16k;
gzip_http_version 1.1;
gzip_comp_level 2;
gzip_types text/plain application/x-javascript text/css application/xml;
gzip_vary on;
```

上述代码中每个参数的含义如下。

- ❑ `gzip`：用于设置开启或者关闭 `gzip` 模块，“`gzip on`”表示开启 GZIP 压缩，实时压缩输出数据流。
- ❑ `gzip_min_length`：设置允许压缩的页面最小字节数，页面字节数从 `header` 头的 `Content-Length` 中获取。默认值是 0，不管页面多大都进行压缩。建议设置成大于 1KB 的字节数，小于 1KB 可能会越压缩越大。
- ❑ `gzip_buffers`：表示申请 4 个单位为 16KB 的内存作为压缩结果流缓存，默认是申请与原始数据大小相同的内存空间来存储 GZIP 压缩结果。
- ❑ `gzip_http_version`：用于设置识别 HTTP 协议版本，默认是 1.1，目前大部分浏览器已经支持 GZIP 解压，使用默认即可。
- ❑ `gzip_comp_level`：用来指定 GZIP 压缩比，1 表示压缩比最小，处理速度最快；9 表示压缩比最大，传输速度快，但处理速度最慢，也比较消耗 CPU 资源。
- ❑ `gzip_types`：用来指定压缩的类型，无论是否指定，“`text/html`”类型总是会被压缩的。
- ❑ `gzip_vary`：让前端的缓存服务器缓存经过 GZIP 压缩的页面，例如用 Squid 缓存经过

Nginx 压缩的数据。

下面这段代码是虚拟主机的配置：

```
server{
listen          80;
server_name     192.168.12.188 www.ixdba.net;
index index.html index.htm index.jsp;
root /web/wwwroot/www.ixdba.net
charset gb2312;
access_log logs/www.ixdba.net.access.log main;
```

建议将对虚拟主机进行配置的内容写在另外一个文件，然后通过 `include` 命令包含进来，这样更便于维护和管理。配置虚拟主机代码中每个参数的含义如下：

- ❑ `server`：定义虚拟主机开始的关键字。
- ❑ `listen`：用于指定虚拟主机的服务端口。
- ❑ `server_name`：用来指定 IP 地址或域名，多个域名之间用空格分开。
- ❑ `index`：用于设定访问的默认首页地址。
- ❑ `root`：用于指定虚拟主机的网页根目录，这个目录可以是相对路径，也可以是绝对路径。
- ❑ `charset`：用于设置网页的默认编码格式。
- ❑ `access_log`：用来指定此虚拟主机的访问日志存放路径，最后的 `main` 用于指定访问日志的输出格式。

下面这段代码是 URL 地址匹配设置：

```
location ~ /\.*(gif|jpg|jpeg|png|bmp|swf)$ {
    root /web/wwwroot/www.ixdba.net;
    expires 30d;
}
```

URL 地址匹配是 Nginx 配置中最灵活的部分。通过 `location` 关键字定义了地址匹配的 begin。location 支持正则表达式匹配，也支持条件判断匹配。用户可以通过 `location` 命令实现 Nginx 对动态、静态网页进行过滤处理。

在上面这段 `location` 代码中，所有扩展名以 `.gif`、`.jpg`、`.jpeg`、`.png`、`.bmp`、`.swf` 结尾的静态文件都交给 Nginx 处理，而 `expires` 用来指定静态文件的过期时间，这里是 30 天。

同理，下面这段代码是将 `upload` 和 `html` 下所有文件都交给 Nginx 来处理，当然，`upload` 和 `html` 目录包含在 `/web/wwwroot/www.ixdba.net` 目录中。

```
location ~ ^/(upload|html)/ {
    root /web/wwwroot/www.ixdba.net;
    expires 30d;
}
```


在下面这段代码中，通过 `location` 关键字实现了将所有以 `.jsp` 为后缀的文件都交给本机的 8080 端口处理。

```
location ~ /\.jsp$ {
    index index.jsp;
    proxy_pass http://localhost:8080;
}
```

下面这段代码配置的是 Nginx 的运行状态，`StubStatus` 模块能够获取 Nginx 自上次启动以来的工作状态：

```
location /NginxStatus {
    stub_status          on;
    access_log            logs/NginxStatus.log;
    auth_basic            "NginxStatus";
    auth_basic_user_file  ../htpasswd;
}
```

上述代码中每个参数的含义如下。

❑ `stub_status`：设置为“on”表示启用 `StubStatus` 的工作状态统计功能。

❑ `access_log`：用来指定 `StubStatus` 模块的访问日志文件。

❑ `auth_basic`：是 Nginx 的一种认证机制。

❑ `auth_basic_user_file`：用来指定认证的密码文件。

由于 Nginx 的 `auth_basic` 认证采用的是与 Apache 兼容的密码文件，因此需要用 Apache 的 `htpasswd` 命令来生成密码文件，例如，要添加一个 `webadmin` 用户，可以使用下面的方式生成密码文件：

```
/usr/local/apache/bin/htpasswd -c /usr/local/nginx/conf/htpasswd webadmin
```

会得到以下提示信息：

```
New password:
```

在输入密码之后，系统会要求再次输入密码。确认密码正确之后成功添加用户。

要查看 Nginx 的运行状态，可以输入 `http://ip/ NginxStatus`，然后输入刚刚创建的用户名和密码，可以看到如下信息：

```
Active connections: 1
server accepts handled requests
 393411 393411 393799
Reading: 0 Writing: 1 Waiting: 0
```

其中，“Active connections”表示当前活跃的连接数，第三行的三个数字表示 Nginx 当

前总共处理了 393 411 个连接，成功创建 393 411 次握手，总共处理了 393 799 个请求。最后一行的“Reading”表示 Nginx 读取到客户端 Header 信息数，“Writing”表示 Nginx 返回给客户端的 Header 信息数，“Waiting”表示 Nginx 已经处理完，正在等候下一次请求命令时的驻留连接数。

最后，设置虚拟主机的错误信息返回页面，代码如下：

```
error_page 404                /404.html;
error_page 500 502 503 504 /50x.html;
location = /50x.html {
    root    html;
}
}
```

通过 error_page 命令可以定制各种错误信息的返回页面。在默认情况下，Nginx 会在主目录的 html 目录中查找指定的返回页面。需要特别注意的是，这些错误信息的返回页面的大小一定要超过 512KB，否则会被 IE 浏览器替换为 IE 默认的错误页面。

10.3.3 Nginx 日常维护技巧

1. Nginx 配置正确性检查

Nginx 提供的配置文件调试功能非常有用，可以快速定位配置文件存在的问题。执行如下命令检测配置文件的正确性：

```
/usr/local/nginx/sbin/nginx -t
```

或者

```
/usr/local/nginx/sbin/nginx -t -c /usr/local/nginx/conf/nginx.conf
```

其中，“-t”参数用于检查配置文件是否正确，但并不执行。“-c”参数用于指定配置文件路径，如果不指定配置文件路径，Nginx 默认会在安装时指定的安装目录下查找 conf/nginx.conf 文件。

如果检测结果显示如下信息，说明配置文件正确。

```
the configuration file /usr/local/nginx/conf/nginx.conf syntax is ok
configuration file /usr/local/nginx/conf/nginx.conf test is successful
```

另外，Nginx 也提供了查看版本及相关编译信息的功能，在命令行执行以下命令可以显示安装 Nginx 的版本信息。

```
/usr/local/nginx/sbin/nginx -v
```

执行以下命令显示安装的 Nginx 版本和相关编译信息：

```
/usr/local/nginx/sbin/nginx -V
```

不但显示 Nginx 的版本信息，同时显示 Nginx 在编译时指定的相关模块信息。

2. Nginx 的启动、关闭与重启

Nginx 对进程的控制能力非常强大，可以通过信号命令控制进程。常用的信号有：

❑ QUIT，表示处理完当前请求后关闭进程。

❑ HUP，表示重新加载配置，即关闭原有的进程并开启新的工作进程。此操作不会中断用户的访问请求，因此，可以通过此信号平滑地重启 Nginx。

❑ USR1，用于 Nginx 的日志切换，也就是重新打开一个日志文件，例如，每天要生成一个新的日志文件时，可以使用这个信号来控制。

❑ USR2，用于平滑升级可执行程序。

❑ WINCH，从容关闭工作进程。

Nginx 的启动非常简单，只需输入：

```
/usr/local/nginx/sbin/nginx
```

即可完成 Nginx 的启动。在 Nginx 启动后，可以通过如下命令查看 Nginx 的启动进程：

```
[root@localhost logs]# ps -ef|grep nginx
root      16572      1  0 11:14 ?        00:00:00 nginx: master process /usr/local/
nginx/sbin/nginx
nobody    16591 16572   0 11:15 ?        00:00:00 nginx: worker process
nobody    16592 16572   0 11:15 ?        00:00:00 nginx: worker process
nobody    16593 16572   0 11:15 ?        00:00:00 nginx: worker process
nobody    16594 16572   0 11:15 ?        00:00:00 nginx: worker process
```

如果要关闭 Nginx 进程，可以使用如下命令：

```
kill -XXX pid
```

其中，“XXX”就是信号名，“pid”是 Nginx 的进程号，可以通过如下两个命令获取：

```
ps -ef | grep "nginx: master process" | grep -v "grep" | awk -F ' ' '{print $2}'
cat /usr/local/nginx/logs/nginx.pid
```

要不间断服务平滑地重新启动 Nginx，可以使用如下命令：

```
kill -HUP 'cat /usr/local/nginx/logs/nginx.pid'
```

10.4 Nginx 常用功能介绍

Nginx 在功能实现方面和性能方面都表现得非常卓越，完全可以与 Apache 相媲美，几乎可以实现 Apache 的所有功能。下面就介绍 Nginx 常用的一些功能实例，具体包含反向代理应用实例、URL 重写应用实例等。

10.4.1 Nginx 反向代理应用实例

反向代理（Reverse Proxy）方式是指通过代理服务器来接受 Internet 上的连接请求，然后将请求转发给内部网络上的服务器，并且将从内部网络服务器上得到的结果返回给 Internet 上请求连接的客户端，此时代理服务器对外就表现为一台服务器。当一台代理服务器能够代理外部网络上的访问请求来访问内部网络时，这种代理服务的方式称为反向代理服务。反向代理服务经常用于 Web 服务器，此时代理服务器在外部网络看来就是一台 Web 服务器，而实际上反向代理服务器并没有保存任何网页的真实数据，所有的静态网页或动态程序都保存在内部网络的 Web 服务器上。因此，对反向代理服务器的攻击并不会使 Web 网站数据遭到破坏，这在一定程度上增强了 Web 服务器的安全性。

反向代理服务器通常也称为 Web 服务加速器，此时反向代理服务器就具有了代理缓存的功能，也就是说，反向代理服务器在接收客户端的请求后，首先从源服务器（内部网络上的 Web 服务器）上获取内容，然后把内容返回给用户，同时，也会把内容保存到代理服务器上一份，这样日后再收到同样的信息请求时，它会把本地缓存里的内容直接发给用户，以此减少后端 Web 服务器的压力，提高响应速度。这其实就是缓存服务器所实现的功能。

1. 多域名跳转应用实例

多域名跳转的需求在 Web 应用配置中经常会用到，Apache 虽然也可实现类似功能，但是实现比较复杂，而通过 Nginx 则可轻松实现。

这里列举一个简单的应用实例。假定一个网站有两个域名，分别是 www.tb.com 和 m.tb.com，要实现当用户访问域名 www.tb.com 时将请求通过 Nginx 代理到 192.168.66.90 的 8080 端口的 web 目录下，而当管理员访问 http://www.tb.com/admin 时将请求代理到 192.168.66.90 的 8080 端口的 /admin 目录下，而当用户访问 m.tb.com 域名时将请求代理到 192.168.66.90 的 8080 端口的 /wap 目录下，只需在 Nginx 配置文件中做如下设置即可。

为了重点说明反向代理的配置，这里仅列出了 Nginx 配置文件中的 server 配置段，而省略了其他部分的配置，特此说明。关于反向代理的配置参数如下：

```
server www.tb.com
location / {
```



```

proxy_pass http://192.168.66.90:8080/web/;
}
location /admin {
proxy_pass http://192.168.66.90:8080/admin;
}
server m.tb.com
location / {
proxy_pass http://192.168.66.90:8080/wap/;
}
}

```

这里要特别注意的是，在通过 proxy_pass 配置的两个代理目录 web 和 wap 后面必须加上一个斜杠，即 “/web/” 和 “/wap/”，否则 Nginx 会出错，仔细看看上面代理配置中两种写法的区别就清楚了。

2. 通过 Nginx 重定向实现新旧域名过渡

在 Web 应用中，需要网页重定向的情况有很多种，例如网页目录结构变动、网页的扩展名改变、网站域名改变、网页重命名等。当因为某种需求，要将网站老的域名用新的域名进行替换，但同时为了不丢失来自搜索引擎的流量，旧的域名也需要在一段时间内能够访问，并且要实现在用旧的域名访问网站时自动将访问请求转到新的域名上，这个时候就需要配置新旧域名过渡。在 Nginx 下配置域名过渡的方法非常简单，有两种方法可实现，下面分别介绍。

这里假定 www.taob.com 为旧的域名，而 www.tb.com 为新的域名，要实现通过 www.taob.com 域名访问的 url 都自动转向到 www.tb.com 对应的 url 路径下，在这里是，当用户访问 “www.taob.com/a/xinwen/2014/0413/2.html” 时，url 地址将自动转向到 “www.tb.com/a/xinwen/2014/0413/2.html” 路径下，第一种设置方法如下：

```

server {
server_name www.taob.com;
rewrite ^/(.*)$ http://www.tb.com/$1 permanent;
}

```

这里仅列出了 Nginx 配置文件中的 server 配置段，而省略了其他部分的内容，这个配置中使用了 Nginx 的重定向功能，通过 rewrite 模块的 permanent 参数实现永久重定向的 HTTP 状态 301。

另外也可以通过判断 Nginx 核心变量 host 实现重定向功能，设置方法如下：

```

server {
server_name www.tb.com www.taob.com;
if ($host != 'www.tb.com') {
rewrite ^/(.*)$ http://www.tb.com/$1 permanent;
}
}

```

这个方法其实也借助了 Nginx 的核心变量 `host` 进行判断, 然后通过 `rewrite` 模块的重定向功能实现转向。

3. alias 在 Nginx 下的应用

Nginx 也可以通过 `alias` 实现类似于 Apache 的别名功能, 例如下面的一个设置:

```
location /i {
    alias /var/www/html/images/;
}
```

在这个 `location` 配置段中, 如果 url 请求 “/i/logo.gif”, 那么 Nginx 将会在服务器上查找 “/var/www/html/images/logo.gif” 文件, 也就是说请求的 url 中 `location` 后面的部分会被追加到 `alias` 指定的目录后面, 而 `location` 后面的 “/i” 路径将被自动丢弃, 类似的例子还有:

```
location ~ ^/download/(.*)$ {
    alias /home/webdata/www/$1;
}
```

在这个例子中, 如果 url 请求 “/download/ebook.tar.gz”, 那么 Nginx 将会在服务器上请求 “/home/webdata/www/ebook.tar.gz” 文件。

在 Nginx 中, `alias` 命令与 `root` 命令所实现的功能有几分相似, 但是也有差别, 例如:

```
location /i {
    root /var/www/html/images/;
}
```

在这个 `location` 配置段中, 如果 url 请求 “/i/logo.gif”, 那么 Nginx 将会在服务器上查找 “/var/www/html/images/i/logo.gif” 文件。同理, 再看下面这个例子:

```
location ~ ^/download/(.*)$ {
    root /home/webdata/www/$1;
}
```

在这个例子中, 如果 url 请求 “/download/ebook.tar.gz”, 那么 Nginx 将会在服务器上查找 “/home/webdata/www/ebook.tar.gz/download/ebook.tar.gz” 文件。

从这两个例子可以看出 `alias` 命令和 `root` 命令的区别: `alias` 指定的目录是当前目录, 而 `root` 指定的是根目录, 一般情况下, 建议在 “`location /`” 中通过 `root` 命令来配置根目录, 而在其他目录匹配的位置使用 `alias` 命令。

4. location 命令应用实例

`location` 命令在 Nginx 配置中最常见, 也最灵活。通过 `location` 命令能完成 Web 系统中

的各种特殊功能和配置。下面介绍 location 命令常见的匹配规则以及规则的优先级，首先看下面四个例子：

```
location = / {
    [ config A ]
}
location / {
    [ config B ]
}
location ^~ /images/ {
    [ config C ]
}
location ~* \.(gif|jpg|jpeg|swf)$ {
    [ config D ]
}
```

在这四个例子中，第一个例子表示只匹配对“/”目录的查询，优先级最高，其他三个例子中的匹配优先级依次降低。第二个例子匹配以“/”开始的所有查询，即所有查询都匹配。第三个例子匹配以“/images/”开始的查询，不再检查正则表达式。第四个例子匹配以 gif、jpg、jpeg、swf 结尾的文件，但匹配优先级低于第三个例子。

Nginx 功能强大，它几乎可以实现 Apache 所有的功能，下面简单介绍 Nginx 中目录权限控制、IP 访问控制、文件访问权限控制如何实现。

要实现与 Apache 一样的目录列表，可添加如下配置：

```
location / {
    autoindex on;
}
```

要实现 IP 访问控制，需要使用 ngx_http_access_module 模块，此模块可以限制某些 IP 地址的客户端访问，看下面一个例子：

```
location / {
    deny 192.168.66.80;
    allow 192.168.66.0/24;
    allow 192.16.88.0/16;
    deny all;
}
```

在这些规则中，Nginx 将按照顺序依次进行规则检测，直到匹配到一条规则，然后退出。

要禁止访问某个目录，可添加如下设置：

```
location ~ ^/(WEB-INF)/ {
```

```
deny all;
```

要禁止访问 *.txt、*.doc 文件，可添加如下设置：

```
location ~* \.(txt|doc){
    root /data/www/wwwroot;
    deny all;
}
```

通过这些介绍可以发现，Nginx 非常灵活，功能强大，完全可以替换 Apache，是一台高性能的 Web 服务器。

10.4.2 Nginx 的 URL 重写应用实例

Nginx 的 URL 重写模块是用得比较多的模块之一，因此这里单独讲述。常用的 URL 重写模块命令有 if、rewrite、set、break 等，下面分别讲述。

1. if 命令

if 命令用于判断一个条件，如果条件成立，则执行后面大括号内的语句，相关配置从上级继承。if 命令的使用方法如下：

语法：if (condition) { ... }

默认值：none

使用字段：server、location

默认情况下，if 命令默认值为空，可在 Nginx 配置文件的 server、location 部分使用，另外，if 命令可以在判断语句中指定正则表达式或匹配条件等，相关匹配条件如下：

正则表达式匹配：

□ ~ 表示区分大小写匹配。

□ ~* 表示不区分大小写匹配。

□ !~ 和 !~* 分别表示区分大小写不匹配以及不区分大小写不匹配。

文件及目录匹配：

□ -f 和 !-f 用来判断是否存在文件。

□ -d 和 !-d 用来判断是否存在目录。

□ -e 和 !-e 用来判断是否存在文件或目录。

□ -x 和 !-x 用来判断文件是否可执行。

Nginx 配置文件中有很多内置变量，这些变量经常和 if 命令一起使用。常见的内置变量有如下几种：

- \$args, 此变量与请求行中的参数相等。
- \$document_root, 此变量等同于当前请求的 root 命令指定的值。
- \$uri, 此变量等同于当前 request 中的 URI。
- \$document_uri, 此变量与 \$uri 含义一样。
- \$host, 此变量与请求头部中“Host”行指定的值一致。
- \$limit_rate, 此变量用来设置限制连接的速率。
- \$request_method, 此变量等同于 request 的 method, 通常是“GET”或“POST”。
- \$remote_addr, 此变量表示客户端 IP 地址。
- \$remote_port, 此变量表示客户端端口。
- \$remote_user, 此变量等同于用户名, 由 ngx_http_auth_basic_module 认证。
- \$request_filename, 此变量表示当前请求的文件的完整路径名, 由 root 或 alias 与 URI request 组合而成。
- \$request_uri, 此变量表示含有参数的完整的初始 URI。
- \$query_string, 此变量与 \$args 含义一致。
- \$server_name, 此变量表示请求到达的服务器名。
- \$server_port, 此变量表示请求到达的服务器的端口号。

在了解完相关的 if 命令规则和 Nginx 内置变量后, 下面给出一个 server 配置段实例:

```
server {
    listen      80;
    server_name www.tb.com;
    access_log  logs/host.access.log  main;
    location / {
        root    /var/www/html;
        index   index.html index.htm;
    }
    location ~*\.(gif|jpg|jpeg|png|bmp|swf|htm|html|css|js)$ {
        root    /usr/local/nginx/www/img;
        if (!-f $request_filename)
        {
            root    /var/www/html/img;
        }
        if (!-f $request_filename)
        {
            root    /apps/images;
        }
    }
    location ~*\.(jsp)$ {
        root    /webdata/webapp/www/ROOT;
        if (!-f $request_filename)
```

```

    {
        root    /usr/local/nginx/www/jsp;
    }
    proxy_pass http://127.0.0.1:8888;
}

```

这段代码主要完成对 `www.tb.com` 这个域名的资源访问配置，`www.tb.com` 这个域名的根目录为 `/var/www/html`，而静态资源分别位于 `/usr/local/nginx/www/img`、`/var/www/html/img`、`/apps/images` 三个目录下，请求静态资源的方式是依次在这三个目录下查找，如果在第一个目录下找不到就找第二个目录，以此类推，如果都找不到，将提示 404 错误。

动态资源分别位于 `/webdata/webapp/www/ROOT` 和 `/usr/local/nginx/www/jsp` 两个目录下，如果客户端请求的资源是以 `.jsp` 结尾的文件，那么将依次在这两个动态程序目录下查找资源。而对于没有在这两个目录中定义的资源，将全部从根目录 `/var/www/html` 进行查找。

2. rewrite 命令

Nginx 通过 `ngx_http_rewrite_module` 模块支持 URL 重写和 if 条件判断，但要使用 `rewrite` 功能，需要 `pcre` 支持，应在编译 Nginx 时指定 `pcre` 源码目录。`rewrite` 的使用语法如下：

语法：rewrite regex flag

默认值：none

使用字段：server、location、if

在默认情况下，`rewrite` 命令默认值为空，可在 Nginx 配置文件的 `server`、`location`、`if` 部分使用，`rewrite` 命令的最后一项参数为 flag 标记，其支持的 flag 标记主要有以下几种：

- last，相当于 Apache 里的 [L] 标记，表示完成 rewrite 之后搜索相应的 URI 或 location。
- break，表示终止匹配，不再匹配后面的规则。
- redirect，将返回 302 临时重定向，在浏览器地址栏会显示跳转后的 URL 地址。
- permanent，将返回 301 永久重定向，在浏览器地址栏会显示跳转后的 URL 地址。

其中，last 和 break 用来实现 URL 重写，浏览器地址栏中的 URL 地址不变。下面是一个示例配置：

```

location ~ ^/best/ {
    rewrite ^/best/(.*)$ /test/$1 break;
}

```

```
proxy_pass http://www.taob.com;
```

这个例子使用了 `break` 标记, 可实现将请求为 `http://www.tb.com/best/webinfo.html` 的页面重定向到 `http://www.taob.com/test/webinfo.html` 页面而不引起浏览器地址栏中 URL 的变化。这个功能在新旧网站交替的时候非常有用。

3. set 命令

通过 `set` 命令可以设置一个变量并为其赋值, 其值可以是文本、变量或它们的组合。也可以使用 `set` 定义一个新的变量, 但是不能使用 `set` 设置 `$http_xxx` 头部变量的值。

`set` 的使用语法如下:

语法: `set variable value`

默认值: `none`

使用字段: `server`、`location`、`if`

在默认情况下, `set` 命令默认值为空, 可在 Nginx 配置文件的 `server`、`location`、`if` 部分使用, 下面是一个示例配置:

```
location / {
    proxy_pass http://127.0.0.1:8080/;
    set $query $query_string;
    rewrite /dede /wordpress?$query;
}
```

在这个例子中, 要实现将请求为 `http://www.tb.com/wp/?p=160` 的页面重写到地址 `http://www.tb.com/wordpress/?p=160`, 也就是重写带参数的 URL。这里涉及 `$query_string` 变量, 这个变量相当于请求行中的参数, 也就是“?”后面的内容, 也可以用 `$args` 代替 `$query_string` 变量。

4. break 命令

`break` 的用法在前面的介绍中其实已经出现过, 它表示完成当前设置的规则后, 不再匹配后面的重写规则。`break` 的使用语法如下:

语法: `break`

默认值: `none`

使用字段: `server`、`location`、`if`

在默认情况下, `break` 命令的值为空, 可在 Nginx 配置文件的 `server`、`location`、`if` 部分使用, 下面是一个应用实例:

```
server {
```

```
listen      80;
server_name www.tb.cn www.taob.com;

if ($host != 'www.tb.cn') {
    rewrite ^/(.*)$ http://www.tb.cn/error.txt
    break;
    rewrite ^/(.*)$ http://www.tb.cn/$1 permanent;
}
```

这个例子定义了两个域名 `www.tb.cn` 和 `www.taob.com`，当通过域名 `www.taob.com` 访问网站时，会将请求重定向到 `http://www.tb.cn/error.txt` 页面，由于设置了 `break` 命令，因此下面的 `rewrite` 规则不再执行，直接退出。

10.5 案例：Nginx 作为 Web 缓存服务器应用

从 0.7.48 版本开始，Nginx 支持类似 Squid 的缓存功能。Nginx 的 Web 缓存服务主要由 `proxy_cache` 相关命令集和 `fastcgi_cache` 相关命令集构成，前者用于反向代理时对后端内容源服务器进行缓存，后者主要用于对 FastCGI 的动态程序进行缓存。此外，如果不想使用 Nginx 自带的缓存功能，也可以使用第三方模块 `ngx_slowfs_cache` 来实现缓存服务器配置。

10.5.1 在 Nginx 下安装缓存服务器

这里使用 Nginx 自带的缓存模块，通过 `proxy_cache` 命令来实现数据的缓存。Nginx 缓存服务器的安装过程基本与 10.2 节介绍的 Nginx 的安装过程一样，唯一不同的是，要在编译 Nginx 的时候加上一个 `ngx_cache_purge` 模块，这个第三方模块是清理 Nginx 缓存的一个插件。下面简单介绍下 Nginx 作为缓存服务器的安装步骤。

`pcrc` 的安装不再重复，首先从 http://labs.frickle.com/nginx_ngx_cache_purge/ 下载 `ngx_cache_purge` 插件，这里下载的文件是 `ngx_cache_purge-2.1.tar.gz`，然后解压即可，过程如下：

```
[root@ngxserver app]# wget http://labs.frickle.com/files/nginx_cache_purge-2.1.tar.gz
[root@ngxserver app]# tar zxvf -C /app/nginx_cache_purge-2.1.tar.gz
```

其次，开始编译安装 Nginx，过程如下：

```
[root@ngxserver app]# tar zxvf nginx-1.4.7.tar.gz
[root@ngxserver app]# cd nginx-1.4.7/
[root@ngxserver app]# ./configure --user=www --group=www --prefix=/usr/local/nginx \
> --add-module=/app/nginx_cache_purge-2.1
> --with-http_stub_status_module --with-http_ssl_module
[root@ngxserver app]# make
```



```
[root@ngxserver app]# make install
```

最后，在完成安装后可以通过“`/usr/local/nginx/sbin/nginx -V`”命令查看已安装 Nginx 的版本和加载的模块信息。

10.5.2 配置 Nginx 缓存服务器

Nginx 缓存服务器的配置主要通过 `proxy_cache` 相关命令集来实现。下面给出 `nginx.conf` 文件的内容，重点说明与缓存相关的参数，其他参数不再重复介绍。

`/usr/local/nginx/conf/nginx.conf` 文件内容如下：

```
user www www;
worker_processes 8;

error_log /usr/local/nginx/logs/nginx_error.log crit;
pid /usr/local/nginx/nginx.pid;
worker_rlimit_nofile 65535;

events
{
    use epoll;
    worker_connections 65535;
}

http
{
    include mime.types;
    default_type application/octet-stream;

    charset utf-8;

    server_names_hash_bucket_size 128;
    client_header_buffer_size 32k;
    large_client_header_buffers 4 32k;
    client_max_body_size 300m;

    sendfile on;
    tcp_nopush on;

    keepalive_timeout 60;

    tcp_nodelay on;

    client_body_buffer_size 512k;
    proxy_connect_timeout 5;
```

```

proxy_read_timeout      60;
proxy_send_timeout      5;
proxy_buffer_size       16k;
proxy_buffers            4 64k;
proxy_busy_buffers_size 128k;
proxy_temp_file_write_size 128k;

```

```

gzip on;
gzip_min_length 1k;
gzip_buffers 4 16k;
gzip_http_version 1.1;
gzip_comp_level 2;
gzip_types text/plain application/x-javascript text/css application/xml;
gzip_vary on;

```

```

proxy_cache_path /backup/proxy_cache_dir levels=1:2 keys_zone=cache_one:4096m
inactive=1d max_size=3g; # proxy_cache_path: 用于设置缓存的目录, 后面跟缓存路径。最好将缓存目
                          # 录放在一个独立的硬盘上
                          # levels=1:2: levels 用来设置目录深度, 这里是两层目录深度, 第一层是一
                          # 个字符, 第二层是两个字符
                          # keys_zone: 用来设置 Web 缓存区名称, 这里是 cache_one, 后面的“4096m”
                          # 表示内存缓存空间大小为 4GB
                          # inactive: 表示自动清除缓存文件的时间, 这里的“1d”表示 1 天没有被访
                          # 问的内容自动清除, 还可以使用分钟或小时计时, 例如“5m”表示 5 分钟后自
                          # 动清除, “5h”表示 5 小时后自动清除
                          # max_size: 表示硬盘缓存空间可使用的最大值, 默认情况下经访问的文件常将
                          # 被放到内存中进行缓存, 而在内存缓存空间不足时, Nginx 会将不经常访问的
                          # 数据从内存写到磁盘

```

```

proxy_temp_path /backup/proxy_temp_dir; # proxy_temp_path 用于指定临时缓存文件的存储路
                                          # 径, 这里需要注意的是, proxy_temp_path 和
                                          # proxy_cache_path 指定的路径必须在同一磁盘分区

```

```

server {
    listen      80;
    server_name www.tb.com www.taob.com;
    charset UTF8;
    access_log logs/cms.access.log main;

```

```

    location / {
        proxy_cache cache_one; # 反向代理缓存设置命令, 语法为“proxy_cache zone|off”,
                                # 默认值为 off, 需要将 proxy_cache 命令放到 location
                                # 字段, 这样匹配此 location 的 url 才能被缓存
        proxy_cache_valid 200 304 12h; # 对不同的 HTTP 状态码设置不同的缓存时间
    }

```

```

proxy_cache_key $host$uri$sis_args$args; # 这个命令是设置以什么样的参数得到缓存的
# 文件名，默认为“#$scheme$proxy_host
# $request_uri”，表示以协议、主机名、
# 请求 uri（包含参数）作 MD5 得出缓存的
# 件文名。这里是以域名、URI、参数组合成
# Web 缓存的 Key 值，Nginx 根据 Key 值哈
# 希，存储缓存内容到二级缓存目录内

proxy_set_header Host $host;
proxy_set_header X-Forwarded-For $remote_addr;
proxy_pass http://127.0.0.1:8080;
expires 1d;
}

# 下面这段用于配置手动清除缓存策略，清除的方法为：如果一个 URL 为
# http://www.tb.com/2014/0413/3.html，那么通过访问
# http://www.tb.com/purge/2014/0413/3.html 即可清除该 URL 的缓存
location ~ /purge(/.*)
{
    allow 127.0.0.1; # 表示只允许指定的 IP 或 IP 段才可以清除 URL 缓存
    allow 192.168.88.0/24;
    deny all;
    proxy_cache_purge cache_one $host$1$sis_args$args;
}

location ~ .*\. (jsp|php|jspx)?$ # 设置不做缓存的内容，这里设置扩展名以 .jsp、.php、
# .jspx 结尾的动态应用程序不缓存
{
    proxy_set_header Host $host;
    proxy_set_header X-Forwarded-For $remote_addr;
    proxy_pass http://http://127.0.0.1:8080;
}

access_log off;
}
}

```

10.5.3 测试 proxy_cache 实现的缓存功能

完成所有配置后，启动 Nginx，然后查看 Nginx 的进程信息如下：

```

[root@haproxy-server conf]# ps -ef|grep nginx
root      2665      1  0 13:25 ?        00:00:00 nginx: master process ../sbin/nginx
nobody    2666    2665  0 13:25 ?        00:00:00 nginx: worker process
nobody    2667    2665  0 13:25 ?        00:00:00 nginx: cache manager process

```

从输出可知, Nginx 进程多出了一个“cache manager”进程, 这个进程就是用来管理缓存服务和文件的。

接着通过 `www.tb.com` 访问任意静态网页, 在打开网页后, 关闭网页再次打开同一个网页, 测试访问速度是否加快。同时, 在缓存目录查看是否已经生成缓存文件, 如果已经生成缓存文件, 表明 Nginx 的缓存服务搭建成功。

其实, 还可以修改已经缓存的静态文件的内容, 然后再次访问这个网页, 看能否显示变动的内容, 如果不能显示, 那么表明这个网页是从缓存中读取的, 等待缓存自动清理时间过后, 再次访问这个页面, 看看内容是否有变动, 如果能够显示变动的网页内容, 表明缓存服务器工作正常。

10.5.4 如何清除指定的 URL 缓存

有时候修改了网页内容, 如果不想等到缓存文件过期后自动清理, 还可以通过手动方式清理缓存文件, 清理方式在上面介绍 Nginx 配置文件时已经做过说明, 只需在清除缓存网页的 URI 地址前面增加 `purge` 即可。

例如, 要清除 `http://www.tb.com/cms/a/xinwen/2014/0413/2.html` 这个页面的缓存文件, 只需在浏览器重输入 `http://www.tb.com/purge/cms/a/xinwen/2014/0413/2.html` 地址即可, 成功清理缓存文件的结果如图 10-3 所示。

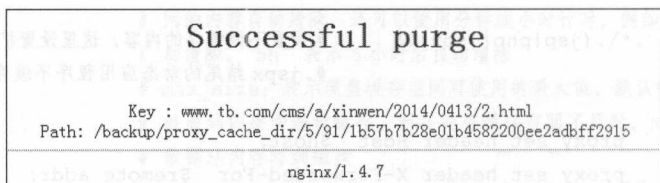


图 10-3 手动清除 Nginx 的 URL 缓存

10.6 案例: Nginx 作为负载均衡服务器应用

Nginx 的负载均衡功能是通过 `upstream` 命令实现的, 因此它的负载均衡机制实现比较简单, 是一个基于内容和应用的 7 层交换负载均衡的实现。Nginx 负载均衡默认对后端服务器有健康检测能力, 但是检测能力较弱, 仅限于端口检测, 在后端服务器比较少的情況下 (10 台以下) 负载均衡能力表现突出。而对于有大量后端节点的负载应用, 由于所有访问请求都从一台服务器进出, 容易发生请求堵塞进而引发连接失败, 因此无法充分发挥后端服务器的性能。

10.6.1 Nginx 的负载均衡算法

Nginx 的负载均衡模块目前支持 4 种调度算法，下面分别进行介绍，其中后两种属于第三方调度方法。

- ❑ 轮询（默认），每个请求按时间顺序逐一分配到不同的后端服务器，如果后端某台服务器死机，自动剔除故障系统，使用户访问不受影响。
- ❑ weight，指定轮询权值，weight 值越大，分配到的访问概率越高，主要用于后端每台服务器性能不均衡的情况下。
- ❑ ip_hash，每个请求按访问 IP 的哈希结果分配，这样来自同一个 IP 的访客固定访问一台后端服务器，有效解决动态网页存在的 session 共享问题。
- ❑ fair，它是比上面两种更加智能的负载均衡算法。此种算法可以依据页面大小和加载时间长短智能地进行负载均衡，也就是根据后端服务器的响应时间来分配请求，响应时间短的优先分配。Nginx 本身是不支持 fair 的，如果需要使用这种调度算法，必须下载 Nginx 的 upstream_fair 模块。
- ❑ url_hash，按访问 URL 的哈希结果来分配请求，使每个 URL 定向到同一台后端服务器，可以进一步提高后端缓存服务器的效率。Nginx 本身是不支持 url_hash 的，如果需要使用这种调度算法，必须安装 Nginx 的 hash 软件包。

在 HTTP Upstream 模块中，可以通过 server 命令指定后端服务器的 IP 地址和端口，同时还可以设定每台后端服务器在负载均衡调度中的状态。常用的状态有：

- ❑ down，表示当前的 server 暂时不参与负载均衡。
- ❑ backup，预留的备份机器。当其他所有的非 backup 机器出现故障或者忙的时候，才会请求 backup 机器，因此这台机器的访问压力最轻。
- ❑ max_fails，允许请求失败的次数，默认为 1。当超过最大次数时，返回 proxy_next_upstream 模块定义的错误。
- ❑ fail_timeout，在经历了 max_fails 次失败后，暂停服务的时间。max_fails 可以和 fail_timeout 一起使用。



注意

当负载调度算法为 ip_hash 时，后端服务器在负载均衡调度中的状态不能是 weight 和 backup。

10.6.2 Nginx 的负载均衡配置实例

下面是一个 Nginx 的负载均衡配置实例，为了重点说明负载均衡的配置，这里仅列出了 Nginx 配置文件中的 http 配置段，而省略了其他部分的配置，特此说明。关于负载均衡

的配置参数如下：

```
http
{
    upstream myserver {
        server 192.168.12.181:80 weight=3 max_fails=3 fail_timeout=20s;
        server 192.168.12.182:80 weight=1 max_fails=3 fail_timeout=20s;
        server 192.168.12.183:80 weight=4 max_fails=3 fail_timeout=20s;
    }

    server
    {
        listen 80;
        server_name www.domain.com 192.168.12.189;
        index index.htm index.html;
        root /ixdba/web/wwwroot;

        location / {
            proxy_pass http://myserver;
            proxy_next_upstream http_500 http_502 http_503 error timeout invalid_header;
            include /opt/nginx/conf/proxy.conf;
        }
    }
}
```

在此段配置中，`upstream` 关键字标识负载均衡配置开始，`upstream` 是 Nginx 的 HTTP Upstream 模块，这个模块通过一个简单的调度算法来实现客户端 IP 到后端服务器的负载均衡。在上面的配置段中，通过 `upstream` 命令指定了一个负载均衡器的名称为 `myserver`。这个名称可以任意指定，在后面需要的地方直接调用即可。

另外，`proxy_next_upstream` 参数用来定义故障转移策略，当后端服务节点返回 500、502、503、504 和执行超时等错误时，自动将请求转发到 `upstream` 负载均衡组中的另一台服务器，实现故障转移。

10.7 Nginx 性能优化技巧

要充分发挥 Nginx 的高效性和稳定性，对 Nginx 的优化非常重要，下面将从编译安装、第三方插件、系统内核等三个方面介绍下如何优化 nginx，最大限度地发挥 Nginx 的高效性。

10.7.1 编译安装过程优化

1. 减小 Nginx 编译后的文件大小

在编译 Nginx 时，默认以 debug 模式进行，而在 debug 模式下会插入很多跟踪和

ASSERT 之类的信息，编译完成后，一个 Nginx 要有好几兆字节。在编译前取消 Nginx 的 debug 模式，编译完成后 Nginx 只有几百千字节，因此，可以在编译之前，修改相关源码，取消 debug 模式，具体方法如下。

在解压 Nginx 源码文件后，找到源码目录下的 auto/cc/gcc 文件，打开然后在其中找到如下几行：

```
# debug
CFLAGS="$CFLAGS -g"
```

注释掉或删除这两行，即可取消 debug 模式。

2. 为特定的 CPU 指定 CPU 类型编译优化

在编译 Nginx 时，默认的 GCC 编译参数是“-O”，要优化 GCC 编译，可以使用以下两个参数：

```
--with-cc-opt='-O3'
--with-cpu-opt=CPU # 为特定的 CPU 编译，有效的值包括：pentium、pentiumpro、pentium3、
# pentium4、athlon、opteron、amd64、sparc32、sparc64、ppc64
```

要确定 CPU 类型，可以通过如下命令实现：

```
[root@localhost home]# cat /proc/cpuinfo | grep "model name"
```

10.7.2 利用 TCMalloc 优化 Nginx 的性能

TCMalloc (Thread-Caching Malloc) 是谷歌开发的开源工具“google-perftools”中的一个成员。与标准的 glibc 库的 malloc 相比，TCMalloc 库在内存分配效率和速度上要高很多，这在很大程度上提高了服务器在高并发情况下的性能，从而降低系统负载。下面简单介绍如何为 Nginx 添加 TCMalloc 库支持。

要安装 TCMalloc 库，需要安装 libunwind (32 位操作系统不需要安装) 和 google-perftools 两个软件包，libunwind 库为基于 64 位 CPU 和操作系统的程序提供了基本函数调用链和函数调用寄存器功能。下面介绍利用 TCMalloc 优化 Nginx 的具体操作过程。

1. 安装 libunwind 库

可以从 <http://download.savannah.gnu.org/releases/libunwind> 下载相应的 libunwind 版本，这里下载的是 libunwind-1.1.tar.gz，安装过程如下：

```
[root@localhost app]# tar zxvf libunwind-1.1.tar.gz
[root@localhost app]# cd libunwind-1.1/
[root@localhost libunwind-1.1]# CFLAGS=-fPIC ./configure
```

```
[root@localhost libunwind-1.1]# make CFLAGS=-fPIC
[root@localhost libunwind-1.1]# make CFLAGS=-fPIC install
```

2. 安装 google-perftools

可以从 <https://code.google.com/p/gperftools> 下载相应的 google-perftools 版本, 这里下载的是 gperftools-2.1.tar.gz, 安装过程如下:

```
[root@localhost app]# tar zxvf gperftools-2.1.tar.gz
[root@localhost app]# cd gperftools-2.1/
[root@localhost gperftools-2.1]# ./configure
[root@localhost gperftools-2.1]# make && make install
[root@localhost gperftools-2.1]# echo "/usr/local/lib" > /etc/ld.so.conf.d/usr_
local_lib.conf
[root@localhost gperftools-2.1]# ldconfig
```

至此, google-perftools 安装完成。

3. 重新编译 Nginx

为了使 Nginx 支持 google-perftools, 需要在安装过程中添加 “--with-google_perftools_module” 选项重新编译 Nginx, 安装代码如下:

```
[root@localhost nginx-1.4.7]# ./configure --prefix=/usr/local/nginx \
> --with-google_perftools_module --with-http_stub_status_module
[root@localhost nginx-1.4.7]# make
[root@localhost nginx-1.4.7]# make install
```

到这里 Nginx 安装完成。

4. 为 google-perftools 添加线程目录

创建一个线程目录, 这里将文件放在 /tmp/tcmalloc 下, 操作如下:

```
[root@localhost app]# mkdir /tmp/tcmalloc
[root@localhost app]# chmod 0777 /tmp/tcmalloc
```

5. 修改 Nginx 主配置文件

修改 nginx.conf 文件, 在 pid 这行的下面添加如下代码:

```
#pid logs/nginx.pid;
google_perftools_profiles /tmp/tcmalloc;
```

接着, 重启 Nginx, 完成 google-perftools 的加载。

6. 验证运行状态

为了验证 google-perftools 已经正常加载，通过如下命令查看：

```
[root@ localhost app]# lsof -n | grep tcmmalloc
nginx      2395 nobody   9w  REG   8,8   0     1599440 /tmp/tcmmalloc.2395
nginx      2396 nobody  11w  REG   8,8   0     1599443 /tmp/tcmmalloc.2396
nginx      2397 nobody  13w  REG   8,8   0     1599441 /tmp/tcmmalloc.2397
nginx      2398 nobody  15w  REG   8,8   0     1599442 /tmp/tcmmalloc.2398
```

由于在 Nginx 配置文件中，设置 worker_processes 的值为 4，因此开启了 4 个 Nginx 线程，每个线程会有一行记录。每个线程文件后面的数字值就是启动的 Nginx 的 PID 值。

至此，利用 TCMalloc 优化 Nginx 的操作完成。

10.7.3 Nginx 内核参数优化

Nginx 内核参数的优化主要是在 Linux 系统中针对 Nginx 应用而进行的系统内核参数优化，常见的优化参数值如下。

下面给出一个优化实例以供参考：

```
net.ipv4.tcp_max_tw_buckets = 6000
net.ipv4.ip_local_port_range = 1024 65000
net.ipv4.tcp_tw_recycle = 1
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_syncookies = 1
net.core.somaxconn = 262144
net.core.netdev_max_backlog = 262144
net.ipv4.tcp_max_orphans = 262144
net.ipv4.tcp_max_syn_backlog = 262144
net.ipv4.tcp_synack_retries = 1
net.ipv4.tcp_syn_retries = 1
net.ipv4.tcp_fin_timeout = 1
net.ipv4.tcp_keepalive_time = 30
```

将上面的内核参数值加入 /etc/sysctl.conf 文件中，然后执行如下命令使之生效：

```
[root@ localhost home]# /sbin/sysctl -p
```

实例中选项的含义如下：

- ❑ net.ipv4.tcp_max_tw_buckets 参数用来设定 timewait 的数量，默认是 180 000，这里设为 6000。
- ❑ net.ipv4.ip_local_port_range 选项用来设定允许系统打开的端口范围。
- ❑ net.ipv4.tcp_tw_recycle 选项用于设置启用 timewait 快速回收。
- ❑ net.ipv4.tcp_tw_reuse 选项用于设置开启重用，允许将 TIME-WAIT sockets 重新用于

新的 TCP 连接。

- `net.ipv4.tcp_syncookies` 选项用于设置开启 SYN cookies，当出现 SYN 等待队列溢出时，启用 cookies 进行处理。
- `net.core.somaxconn` 选项默认值是 128，这个参数用于调节系统同时发起的 TCP 连接数，在高并发的请求中，默认的值可能会导致链接超时或者重传，因此，需要结合并发请求数来调节此值。
- `net.core.netdev_max_backlog` 选项表示当每个网络接口接收数据包的速率比内核处理这些包的速率快时，允许发送到队列的数据包的最大数目。
- `net.ipv4.tcp_max_orphans` 选项用于设定系统中最多有多少个 TCP 套接字不被关联到任何一个用户文件句柄上。如果超过这个数字，孤立连接将立即被复位并输出警告信息。这个限制只是为了防止简单的 DoS 攻击。不能过分依靠这个限制甚至人为减小这个值，更多的情况是增加这个值。
- `net.ipv4.tcp_max_syn_backlog` 选项用于记录那些尚未收到客户端确认信息的连接请求的最大值。对于有 128MB 内存的系统而言，此参数的默认值是 1024，对小内存的系统则是 128。
- `net.ipv4.tcp_synack_retries` 参数的值决定了内核放弃连接之前发送 SYN+ACK 包的数量。
- `net.ipv4.tcp_syn_retries` 选项表示在内核放弃建立连接之前发送 SYN 包的数量。
- `net.ipv4.tcp_fin_timeout` 选项决定了套接字保持在 FIN-WAIT-2 状态的时间。默认值是 60 秒。正确设置这个值非常重要，有时候即使一台负载很小的 Web 服务器，也会出现因为大量的死套接字而产生内存溢出的风险。
- `net.ipv4.tcp_keepalive_time` 选项表示当 keepalive 启用的时候，TCP 发送 keepalive 消息的频度。默认值是 2（单位是小时）。

高性能集群软件 Keepalived

11.1 Keepalived 介绍

Keepalived 是 Linux 下一个轻量级的高可用解决方案，它与 HeartBeat、RoseHA 实现的功能类似，都可以实现服务或者网络的高可用，但是又有差别，HeartBeat 是一个专业的、功能完善的高可用软件，它提供了 HA 软件所需的基本功能，比如心跳检测和资源接管，监测集群中的系统服务，在群集节点间转移共享 IP 地址的所有者等，HeartBeat 功能强大，但是部署和使用相对比较麻烦；与 HeartBeat 相比，Keepalived 主要是通过虚拟路由冗余来实现高可用功能，虽然它没有 HeartBeat 功能强大，但 Keepalived 部署和使用非常简单，所有配置只需一个配置文件即可完成。这也是本章重点介绍 Keepalived 的原因。

11.1.1 Keepalived 是什么

Keepalived 起初是为 LVS 设计的，专门用来监控集群系统中各个服务节点的状态。它根据 TCP/IP 参考模型的第三、第四和第五层交换机制检测每个服务节点的状态，如果某个服务节点出现异常，或工作出现故障，Keepalived 将检测到，并将出现故障的服务节点从集群系统中剔除，而在故障节点恢复正常后，Keepalived 又可以自动将此服务节点重新加入服务器集群中，这些工作全部自动完成，不需要人工干涉，需要人工完成的只是修复出现故障的服务节点。

Keepalived 后来又加入了 VRRP 的功能，VRRP (Virtual Router Redundancy Protocol, 虚拟路由器冗余协议) 出现的目的是解决静态路由出现的单点故障问题，通过 VRRP 可以

实现网络不间断稳定运行。因此, Keepalived 一方面具有服务器状态检测和故障隔离功能, 另一方面也具有 HA cluster 功能。下面详细介绍 VRRP 协议的实现过程。

11.1.2 VRRP 协议与工作原理

在现实的网络环境中, 主机之间的通信都是通过配置静态路由(默认网关)完成的, 而主机之间的路由器一旦出现故障, 通信就会失败, 因此, 在这种通信模式中, 路由器就成了一个单点瓶颈, 为了解决这个问题, 就引入了 VRRP 协议。

熟悉网络的读者对 VRRP 协议应该并不陌生。它是一种主备模式的协议, 通过 VRRP 可以在网络发生故障时透明地进行设备切换而不影响主机间的数据通信, 这其中涉及两个概念: 物理路由器和虚拟路由器。

VRRP 可以将两台或多台物理路由器设备虚拟成一个虚拟路由器, 这个虚拟路由器通过虚拟 IP (一个或多个) 对外提供服务, 而在虚拟路由器内部是多个物理路由器协同工作, 同一时间只有一台物理路由器对外提供服务, 这台物理路由器被称为主路由器(处于 MASTER 角色)。一般情况下 MASTER 由选举算法产生, 它拥有对外服务的虚拟 IP, 提供各种网络功能, 如 ARP 请求、ICMP、数据转发等。而其他物理路由器不拥有对外的虚拟 IP, 也不提供对外网络功能, 仅仅接收 MASTER 的 VRRP 状态通告信息, 这些路由器被统称为备份路由器(处于 BACKUP 角色)。当主路由器失效时, 处于 BACKUP 角色的备份路由器将重新进行选举, 产生一个新的主路由器进入 MASTER 角色继续提供对外服务, 整个切换过程对用户来说完全透明。

每个虚拟路由器都有一个唯一标识, 称为 VRID, 一个 VRID 与一组 IP 地址构成了一个虚拟路由器。在 VRRP 协议中, 所有的报文都是通过 IP 多播形式发送的, 而在一个虚拟路由器中, 只有处于 MASTER 角色的路由器会一直发送 VRRP 数据包, 处于 BACKUP 角色的路由器只接收 MASTER 发过来的报文信息, 用来监控 MASTER 运行状态; 因此, 不会发生 BACKUP 抢占的现象, 除非它的优先级更高。而当 MASTER 不可用时, BACKUP 也就无法收到 MASTER 发过来的报文信息, 于是就认定 MASTER 出现故障, 接着多台 BACKUP 就会进行选举, 优先级最高的 BACKUP 将成为新的 MASTER, 这种选举并进行角色切换的过程非常快, 因而保证了服务的持续可用性。

11.1.3 Keepalived 工作原理

上节简单介绍了 Keepalived 通过 VRRP 实现高可用功能的工作原理, 而 Keepalived 作为一个高性能集群软件, 它还能实现对集群中服务器运行状态的监控及故障隔离。下面继续介绍下 Keepalived 对服务器运行状态监控和检测的工作原理。

Keepalived 工作在 TCP/IP 参考模型的第三、第四和第五层，也就是网络层、传输层和应用层。根据 TCP/IP 参考模型各层所能实现的功能，Keepalived 运行机制如下。

- 在网络层，运行着 4 个重要的协议：互联网协议 IP、互联网控制报文协议 ICMP、地址转换协议 ARP 以及反向地址转换协议 RARP。Keepalived 在网络层采用的最常见的工作方式是通过 ICMP 协议向服务器集群中的每个节点发送一个 ICMP 的数据包（类似于 ping 实现的功能），如果某个节点没有返回响应数据包，那么认为此节点发生了故障，Keepalived 将报告此节点失效，并从服务器集群中剔除故障节点。
- 在传输层，提供了两个主要的协议：传输控制协议 TCP 和用户数据协议 UDP。传输控制协议 TCP 可以提供可靠的数据传输服务、IP 地址和端口，代表 TCP 的一个连接端。要获得 TCP 服务，需要在发送机的一个端口上和接收机的一个端口上建立连接，而 Keepalived 在传输层就是利用 TCP 协议的端口连接和扫描技术来判断集群节点是否正常的。比如，对于常见的 Web 服务默认的 80 端口、SSH 服务默认的 22 端口等，Keepalived 一旦在传输层探测到这些端口没有响应数据返回，就认为这些端口发生异常，然后强制将此端口对应的节点从服务器集群组中移除。
- 在应用层，可以运行 FTP、TELNET、SMTP、DNS 等各种不同类型的高层协议，Keepalived 的运行方式也更加全面化和复杂化，用户可以通过自定义 Keepalived 的工作方式，例如，用户可以通过编写程序来运行 Keepalived，而 Keepalived 将根据用户的设定检测各种程序或服务是否允许正常，如果 Keepalived 的检测结果与用户设定不一致时，Keepalived 将把对应的服务从服务器中移除。

11.1.4 Keepalived 的体系结构

Keepalived 是一个高度模块化的软件，结构简单，但扩展性很强，感兴趣的读者可以阅读 Keepalived 的源码。图 11-1 是官方给出的 Keepalived 体系结构拓扑图。

从图 11-1 中可以看出，Keepalived 的体系结构从整体上分为两层，分别是用户空间层（User Space）和内核空间层（Kernel Space）。下面介绍 Keepalived 两层结构的详细组成及实现的功能。

内核空间层处于最底层，它包括 IPVS 和 NETLINK 两个模块。IPVS 模块是 Keepalived 引入的一个第三方模块，通过 IPVS 可以实现基于 IP 的负载均衡集群。IPVS 默认包含在 LVS 集群软件中。而对于 LVS 集群软件，相信做运维的读者并不陌生：在 LVS 集群中，IPVS 安装在一台叫做 Director Server 的服务器上，同时在 Director Server 上虚拟出一个 IP 地址对外提供服务，而用户必须通过这个虚拟 IP 地址才能访问服务。这个虚拟 IP 一般称为 LVS 的 VIP，即 Virtual IP。访问的请求首先经过 VIP 到达 Director Server，然后由 Director Server 从服务器集群节点中选取一个服务节点响应用户的请求。

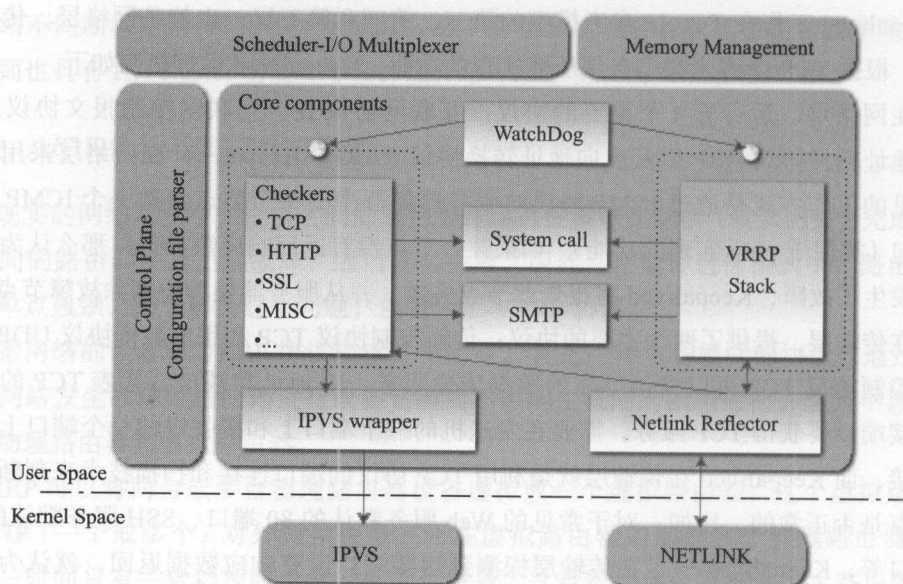


图 11-1 Keepalived 体系结构图

Keepalived 最初就是为 LVS 提供服务的，由于 Keepalived 可以实现对集群节点的状态检测，而 IPVS 可以实现负载均衡功能，因此，Keepalived 借助于第三方模块 IPVS 就可以很方便地搭建一套负载均衡系统。在这里有个误区，由于 Keepalived 可以和 IPVS 一起很好地工作，因此很多初学者都以为 Keepalived 就是一个负载均衡软件，这种理解是错误的。

在 Keepalived 中，IPVS 模块是可配置的，如果需要负载均衡功能，可以在编译 Keepalived 时打开负载均衡功能，也可以通过配置编译参数关闭。

NETLINK 模块主要用于实现一些高级路由框架和一些相关的网络功能，完成用户空间层 Netlink Reflector 模块发来的各种网络请求。

用户空间层位于内核空间层之上，Keepalived 的所有具体功能都在这里实现，下面介绍几个重要部分所实现的功能。

在用户空间层，Keepalived 又分为 4 个部分，分别是 Scheduler I/O Multiplexer、Memory Management、Control Plane 和 Core components。其中，Scheduler I/O Multiplexer 是一个 I/O 复用分发调度器，它负责安排 Keepalived 所有内部的任务请求。Memory Management 是一个内存管理机制，这个框架提供了访问内存的一些通用方法。Control Plane 是 Keepalived 的控制面板，可以实现对配置文件进行编译和解析，Keepalived 的配置文件解析比较特殊，它并不是一次解析所有模块的配置，而是只有在用到某模块时才解析相应的配置。最后详细说一下 Core components，这部分是 Keepalived 的核心组件，包含了一系列功能模块，主要有 WatchDog、Checkers、VRRP Stack、IPVS wrapper 和 Netlink

Reflector, 下面介绍每个模块所实现的功能如下。

(1) WatchDog

WatchDog 是计算机可靠性领域中一个极为简单又非常有效的检测工具, 它的工作原理是针对被监视的目标设置一个计数器和一个阈值, WatchDog 会自己增加此计数值, 然后等待被监视的目标周期性地重置该计数值。一旦被监控目标发生错误, 就无法重置此计数值, WatchDog 就会检测到, 于是采取对应的恢复措施, 例如重启或关闭。

在 Linux 中很早就引入了 WatchDog 功能, 而 Keepalived 正是通过 WatchDog 的运行机制来监控 Checkers 和 VRRP 进程的。

(2) Checkers

这是 Keepalived 最基础的功能, 也是最主要的功能, 可实现对服务器运行状态检测和故障隔离。

(3) VRRP Stack

这是 Keepalived 后来引入的 VRRP 功能, 可以实现 HA 集群中失败切换 (Failover) 功能。Keepalived 通过 VRRP 功能再结合 LVS 负载均衡软件即可部署一个高性能的负载均衡集群系统。

(4) IPVS wrapper

这是 IPVS 功能的一个实现。IPVS wrapper 模块可以将设置好的 IPVS 规则发送到内核空间并提交给 IPVS 模块, 最终实现 IPVS 模块的负载均衡功能。

(5) Netlink Reflector

用来实现高可用集群中 Failover 时虚拟 IP (VIP) 的设置和切换。Netlink Reflector 的所有请求最后都发送到内核空间层的 NETLINK 模块来完成。

11.2 Keepalived 安装与配置

11.2.1 Keepalived 的安装过程

Keepalived 的安装非常简单, 下面通过源码编译的方式介绍 Keepalived 的安装过程。首先打开 Keepalived 的官方网址 <http://www.keepalived.org>, 从中可以下载到各种版本的 Keepalived, 这里下载的是 keepalived-1.2.12.tar.gz。以操作系统环境 CentOS6.3 为例, Keepalived 安装步骤如下:

```
[root@keepalived-master app]# tar zxvf keepalived-1.2.12.tar.gz
[root@keepalived-master app]# cd keepalived-1.2.12
[root@keepalived-master keepalived-1.2.12]# ./configure --sysconf=/etc \
```



```
> --with-kernel-dir=/usr/src/kernels/2.6.32-431.5.1.el6.x86_64
[root@keepalived-master keepalived-1.2.12]# make
[root@keepalived-master keepalived-1.2.12]# make install
[root@keepalived-master keepalived-1.2.12]# ln -s /usr/local/sbin/keepalived /sbin/
[root@keepalived-master keepalived-1.2.12]# chkconfig --add keepalived
[root@keepalived-master keepalived-1.2.12]# chkconfig --level 35 keepalived on
```

在编译选项中，“--sysconf”指定了 Keepalived 配置文件的安装路径，即路径为 /etc/Keepalived/Keepalived.conf；“--with-kernel-dir”是个很重要的参数，但这个参数并不是要把 Keepalived 编译进内核，而是指定使用内核源码中的头文件，即 include 目录。只有在使用 LVS 时，才需要用到“--with-kernel-dir”参数，其他时候是不需要的。

在安装完成后，会得到如图 11-2 所示的内容。

```
Keepalived configuration
-----
Keepalived version      : 1.2.12
Compiler                : gcc
Compiler flags          : -g -O2 -DFALLBACK_LIBNL1
Extra Lib               : -lssl -lcrypto -lcrypt -lnl
Use IPVS Framework      : Yes
IPVS sync daemon support : Yes
IPVS use libnl          : Yes
Use VRRP Framework      : Yes
Use VRRP VMAC           : Yes
SNMP support            : No
SHA1 support            : No
Use Debug flags         : No
```

图 11-2 Keepalived 编译输出模块信息

图 11-2 显示的是 Keepalived 输出的加载模块信息，其中：

- ❑ Use IPVS Framework 表示使用 IPVS 框架，也就是负载均衡模块，后面的“Yes”表示启用 IPVS 功能。一般在搭建高可用负载均衡集群时会启用 IPVS 功能，如果只是使用 Keepalived 的高可用功能，则不需要启用 IPVS 模块，可以在编译 Keepalived 时通过“--disable-lvs”关闭 IPVS 功能。
- ❑ IPVS sync daemon support 表示启用 IPVS 的同步功能，此模块一般和 IPVS 模块一起使用，如果需要关闭，可在编译 Keepalived 时通过“--disable-lvs-syncd”参数实现。
- ❑ IPVS use libnl 表示使用新版的 libnl。libnl 是 NETLINK 的一个实现，如果要使用新版的 libnl，需要在系统中安装 libnl 和 libnl-devel 软件包。
- ❑ Use VRRP Framework 表示使用 VRRP 框架，这是实现 Keepalived 高可用功能必需的模块。
- ❑ Use VRRP VMAC 表示使用基础 VMAC 接口的 xmit VRRP 包，这是 Keepalived 在 1.2.10 版本及以后新增的一个功能。

至此，Keepalived 的安装介绍完毕。下面开始进入 Keepalived 配置的讲解。

11.2.2 Keepalived 的全局配置

在上节安装 Keepalived 的过程中，指定了 Keepalived 配置文件的路径为 /etc/Keepalived/Keepalived.conf，Keepalived 的所有配置均在这个配置文件中完成。由于 Keepalived.conf 文件中可配置的选项比较多，这里根据配置文件所实现的功能，将 Keepalived 配置分为三类，分别是：全局配置（Global Configuration）、VRRPD 配置和 LVS 配置。下面将主要介绍下 Keepalived 配置文件中一些常用配置选项的含义和用法。

Keepalived 的配置文件都是以块（block）的形式组织的，每个块的内容都包含在 {} 中，以“#”和“!”开头的行都是注释。全局配置就是对整个 Keepalived 都生效的配置，基本内容如下：

```
! Configuration File for keepalived
global_defs {
    notification_email {
        dba.gao@gmail.com
        ixdba@163.com
    }
    notification_email_from Keepalived@localhost
    smtp_server 192.168.200.1
    smtp_connect_timeout 30
    router_id LVS_DEVEL
}
```

全局配置以“global_defs”作为标识，在“global_defs”区域内的都是全局配置选项，其中：

- ❑ notification_email 用于设置报警邮件地址，可以设置多个，每行一个。注意，如果要开启邮件报警，需要开启本机的 Sendmail 服务。
- ❑ notification_email_from 用于设置邮件的发送地址。
- ❑ smtp_server 用于设置邮件的 smtp server 地址。
- ❑ smtp_connect_timeout 用于设置连接 smtp server 的超时时间。
- ❑ router_id 表示运行 Keepalived 服务器的一个标识，是发邮件时显示在邮件主题中的信息。

11.2.3 Keepalived 的 VRRPD 配置

VRRPD 配置是 Keepalived 所有配置的核心，主要用来实现 Keepalived 的高可用功能。从结构上来看，VRRPD 配置又可分为 VRRP 同步组配置和 VRRP 实例配置。

这里首先介绍同步组实现的主要功能。同步组是相对于多个 VRRP 实例而言的，在多个 VRRP 实例的环境中，每个 VRRP 实例所对应的网络环境会有所不同，假设一个实例处于网段 A，另一个实例处于网段 B，而如果 VRRPD 只配置了 A 网段的检测，那么当 B 网段主机出现故障时，VRRPD 会认为自身仍处于正常状态，进而不会进行主备节点的切换，这样问题就出现了。同步组就是用来解决这个问题的，将所有 VRRP 实例都加入同步组中，这样任何一个实例出现问题，都会导致 Keepalived 进行主备切换。

下面是两个同步组的配置样例：

```

vrrp_sync_group G1 {
    group {
        VI_1
        VI_2
        VI_5
    }
    notify_backup "/usr/local/bin/vrrp.back arg1 arg2"
    notify_master "/usr/local/bin/vrrp.mast arg1 arg2"
    notify_fault "/usr/local/bin/vrrp.fault arg1 arg2"
    notify_stop "/usr/local/bin/vrrp.stop arg1 arg2"
}

vrrp_sync_group G2 {
    group {
        VI_3
        VI_4
    }
}

```

其中，G1 同步组包含 VI_1、VI_2、VI_5 三个 VRRP 实例，G2 同步组包含 VI_3、VI_4 两个 VRRP 实例。这 5 个实例将在 vrrp_instance 段进行定义。另外，在 vrrp_sync_group 段中还出现了 notify_master、notify_backup、notify_fault 和 notify_stop 四个选项，这是 Keepalived 配置中的一个通知机制，也是 Keepalived 包含的四种状态。下面介绍每个选项的含义。

- notify_master：指定当 Keepalived 进入 MASTER 状态时要执行的脚本，这个脚本可以是一个状态报警脚本，也可以是一个服务管理脚本。Keepalived 允许脚本传入参数，因此灵活性很强。
- notify_backup：指定当 Keepalived 进入 BACKUP 状态时要执行的脚本，同理，这个脚本可以是一个状态报警脚本，也可以是一个服务管理脚本。
- notify_fault：指定当 Keepalived 进入 FAULT 状态时要执行的脚本，脚本功能与前两个类似。
- notify_stop：指定当 Keepalived 程序终止时需要执行的脚本。

下面正式进入 VRRP 实例的配置，也就是配置 Keepalived 的高可用功能。VRRP 实例段主要用来配置节点角色（主或从）、实例绑定的网络接口、节点间验证机制、集群服务 IP 等。下面是实例 VI_1 的一个配置样例。

```
vrrp_instance VI_1 {
    state MASTER
    interface eth0
    virtual_router_id 51
    priority 100
    advert_int 1
    mcast_src_ip <IPADDR>
    garp_master_delay 10

    track_interface {
        eth0
        eth1
    }

    authentication {
        auth_type PASS
        auth_pass qwaszx
    }

    virtual_ipaddress {
        #<IPADDR>/<MASK> brd <IPADDR> dev <STRING> scope <SCOPE> label <LABEL>
        192.168.200.16
        192.168.200.17 dev eth1
        192.168.200.18 dev eth2
    }

    virtual_routes {
        #src <IPADDR> [to] <IPADDR>/<MASK> via|gw <IPADDR> dev <STRING> scope <SCOPE>
        src 192.168.100.1 to 192.168.109.0/24 via 192.168.200.254 dev eth1
        192.168.110.0/24 via 192.168.200.254 dev eth1
        192.168.111.0/24 dev eth2
        192.168.112.0/24 via 192.168.100.254
        192.168.113.0/24 via 192.168.100.252 or 192.168.100.253
    }

    nopreempt
    preempt_delay 300
}
```

以上 VRRP 配置以“vrrp_instance”作为标识，在这个实例中包含了若干配置选项，分别介绍如下。

- ❑ vrrp_instance 是 VRRP 实例开始的标识，后跟 VRRP 实例名称。
- ❑ state 用于指定 Keepalived 的角色，MASTER 表示此主机是主服务器，BACKUP 表示此主机是备用服务器。

- ❑ `interface` 用于指定 HA 监测网络的接口。
- ❑ `virtual_router_id` 是虚拟路由标识，这个标识是一个数字，同一个 VRRP 实例使用唯一的标识，即在同一个 `vrrp_instance` 下，MASTER 和 BACKUP 必须是一致的。
- ❑ `priority` 用于定义节点优先级，数字越大表示节点的优先级就越高。在一个 VRRP_instance 下，MASTER 的优先级必须大于 BACKUP 的优先级。
- ❑ `advert_int` 用于设定 MASTER 与 BACKUP 主机之间同步检查的时间间隔，单位是秒。
- ❑ `mcast_src_ip` 用于设置发送多播包的地址，如果不设置，将使用绑定的网卡所对应的 IP 地址。
- ❑ `garp_master_delay` 用于设定在切换到 MASTER 状态后延时进行 Gratuitous arp 请求的时间。
- ❑ `track_interface` 用于设置一些额外的网络监控接口，其中任何一个网络接口出现故障，Keepalived 都会进入 FAULT 状态。
- ❑ `authentication` 用于设定节点间通信验证类型和密码，验证类型主要有 PASS 和 AH 两种，在一个 `vrrp_instance` 下，MASTER 与 BACKUP 必须使用相同的密码才能正常通信。
- ❑ `virtual_ipaddress` 用于设置虚拟 IP 地址（VIP），又叫做漂移 IP 地址。可以设置多个虚拟 IP 地址，每行一个。之所以称为漂移 IP 地址，是因为 Keepalived 切换到 MASTER 状态时，这个 IP 地址会自动添加到系统中，而切换到 BACKUP 状态时，这些 IP 又会自动从系统中删除。Keepalived 通过 “ip address add” 命令的形式将 VIP 添加进系统中。要查看系统中添加的 VIP 地址，可以通过 “ip add” 命令实现。`virtual_ipaddress` 段中添加的 IP 形式可以多种多样，例如可以写成 “192.168.16.189/24 dev eth1” 这样的形式，而 Keepalived 会使用 IP 命令 “ip addr add 192.168.16.189/24 dev eth1” 将 IP 信息添加到系统中。因此，这里的配置规则和 IP 命令的使用规则是一致的。
- ❑ `virtual_routes` 和 `virtual_ipaddress` 段一样，用来设置在切换时添加或删除相关路由信息。使用方法和例子可以参考上面的示例。通过 “ip route” 命令可以查看路由信息是否添加成功，此外，也可以通过上面介绍的 `notify_master` 选项来代替 `virtual_routes` 实现相同的功能。
- ❑ `nopreempt` 设置的是高可用集群中的不抢占功能。在一个 HA Cluster 中，如果主节点死机了，备用节点会进行接管，主节点再次正常启动后一般会自动接管服务。这种来回切换的操作，对于实时性和稳定性要求不高的业务系统来说，还是可以接受的，而对于稳定性和实时性要求很高的业务系统来说，不建议来回切换，毕竟服务的切

换存在一定的风险和不确定性,在这种情况下,就需要设置 `nopreempt` 这个选项了。设置 `nopreempt` 可以实现主节点故障恢复后不再切回到主节点,让服务一直在备用节点工作,直到备用节点出现故障才会进行切换。在使用不抢占时,只能在“state”状态为 `BACKUP` 的节点上设置,而且这个节点的优先级必须高于其他节点。

❑ `preempt_delay` 用于设置抢占的延时时间,单位是秒。有时候系统启动或重启之后网络需要经过一段时间才能正常工作,在这种情况下进行主备切换是没必要的,此选项就是用来设置这种情况发生的时间间隔。在此时间内发生的故障将不会进行切换,而如果超过 `preempt_delay` 指定的时间,并且网络状态异常,那么才开始进行主备切换。

11.2.4 Keepalived 的 LVS 配置

由于 Keepalived 属于 LVS 的扩展项目,因此,Keepalived 可以与 LVS 无缝整合,轻松搭建一套高性能的负载均衡集群系统。下面介绍 Keepalived 配置文件中关于 LVS 配置段的配置方法。

LVS 段的配置以“`virtual_server`”作为开始标识,此段内容由两部分组成,分别是 `real_server` 段和健康检测段。下面是 `virtual_server` 段常用选项的一个配置示例:

```
virtual_server 192.168.12.200 80 {
    delay_loop 6
    lb_algo rr
    lb_kind DR
    persistence_timeout 50
    persistence_granularity <NETMASK>
    protocol TCP
    ha_suspend
    virtualhost <string>
    sorry_server <IPADDR> <PORT>
```

上述示例中每个选项的含义如下。

❑ `virtual_server`: 是设置虚拟服务器开始的标识,后面跟虚拟 IP 地址和服务端口,IP 与端口之间用空格隔开。

❑ `delay_loop`: 设置健康检查的时间间隔,单位是秒。

❑ `lb_algo`: 设置负载调度算法,可用的调度算法有 `rr`、`wrr`、`lc`、`wlc`、`lbic`、`sh`、`dh` 等,常用的算法有 `rr` 和 `wlc`。

❑ `lb_kind`: 设置 LVS 实现负载均衡的机制,有 `NAT`、`TUN` 和 `DR` 三个模式可选。

❑ `persistence_timeout`: 会话保持时间,单位是秒。这个选项对动态网页是非常有用的,为集群系统中的 session 共享提供了一个很好的解决方案。有了这个会话保持功能,用户的请求会一直分发到某个服务节点,直到超过这个会话的保持时间。需要注意

的是，这个会话保持时间是最大无响应超时时间，也就是说，用户在操作动态页面时，如果在 50s 内没有执行任何操作，那么接下来的操作会被分发到另外的节点，但是如果用户一直在操作动态页面，则不受 50s 的时间限制。

❑ **persistence_granularity**：此选项是配合 **persistence_timeout** 的，后面跟的值是子网掩码，表示持久连接的粒度。默认是 255.255.255.255，也就是一个单独的客户端 IP。如果将掩码修改为 255.255.255.0，那么客户端 IP 所在的整个网段的请求都会分配到同一台 **real server** 上。

❑ **protocol**：指定转发协议类型，有 TCP 和 UDP 两种可选。

❑ **ha_suspend**：节点状态从 MASTER 到 BACKUP 切换时，暂不启用 **real server** 节点的健康检查。

❑ **virtualhost**：在通过 HTTP_GET/SSL_GET 做健康检测时，指定的 Web 服务器的虚拟主机地址。

❑ **sorry_server**：相当于一个备用节点，在所有 **real server** 失效后，这个备用节点会启用。

下面是 **real_server** 段的一个配置示例：

```
real_server 192.168.12.132 80 {
weight 3
inhibit_on_failure
notify_up <STRING> | <QUOTED-STRING>
notify_down <STRING> | <QUOTED-STRING>
}
```

上述示例中每个选项的含义如下。

❑ **real_server**：是设置 **real_server** 段开始的标识，用来指定 **real server** 节点，后面跟的是 **real server** 的真实 IP 地址和端口，IP 与端口之间用空格隔开。

❑ **weight**：用来配置 **real server** 节点的权值。权值大小用数字表示，数字越大，权值越高。设置权值的大小可以为不同性能的服务器分配不同的负载，为性能高的服务器设置较高的权值，而为性能较低的服务器设置相对较低的权值，这样才能合理地利用和分配系统资源。

❑ **inhibit_on_failure**：表示在检测到 **real server** 节点失效后，把它的“weight”值设置为 0，而不是从 IPVS 中删除。

❑ **notify_up**：此选项与上面介绍过的 **notify_maser** 有相同的功能，后跟一个脚本，表示在检测到 **real server** 节点服务处于 UP 状态后执行的脚本。

❑ **notify_down**：表示在检测到 **real server** 节点服务处于 DOWN 状态后执行的脚本。

健康检测段允许多种检查方式，常见的有 HTTP_GET、SSL_GET、TCP_CHECK、SMTP_CHECK、MISC_CHECK。首先看 TCP_CHECK 检测方式示例：

```
TCP_CHECK {
    connect_port 80
    connect_timeout 3
    nb_get_retry 3
    delay_before_retry 3
}
```

上述检测示例中每个选项的含义如下。

❑ **connect_port**: 健康检查的端口, 如果不指定, 默认是 **real_server** 指定的端口。

❑ **connect_timeout**: 表示无响应超时时间, 单位是秒, 这里是 3s 超时。

❑ **nb_get_retry**: 表示重试次数, 这里是 3 次。

❑ **delay_before_retry**: 表示重试间隔, 这里是间隔 3s。

下面是 HTTP_GET 和 SSL_GET 检测方式的示例:

```
HTTP_GET | SSL_GET
{
    url {
        path /index.html
        digest e6c271eb5f017f280cf97ec2f51b02d3
        status_code 200
    }
    connect_port 80
    bindto 192.168.12.80
    connect_timeout 3
    nb_get_retry 3
    delay_before_retry 2
}
```

上述示例中每个选项的含义如下。

❑ **url**: 用来指定 HTTP/SSL 检查的 URL 信息, 可以指定多个 URL。

❑ **path**: 后跟详细的 URL 路径。

❑ **digest**: SSL 检查后的摘要信息, 这些摘要信息可以通过 **genhash** 命令工具获取。例如: **genhash -s 192.168.12.80 -p 80 -u /index.html**。

❑ **status_code**: 指定 HTTP 检查返回正常状态码的类型, 一般是 200。

❑ **bindto**: 表示通过此地址来发送请求对服务器进行健康检查。

下面是 MISC_CHECK 检测方式的示例:

```
MISC_CHECK
{
    misc_path /usr/local/bin/script.sh
    misc_timeout 5
    !misc_dynamic
}
```

MISC 健康检查方式可以通过执行一个外部程序来判断 real server 节点的服务状态，使用非常灵活。以下是常用的几个选项的含义。

- ❑ `misc_path`: 用来指定一个外部程序或者一个脚本路径。
- ❑ `misc_timeout`: 设定执行脚本的超时时间。
- ❑ `misc_dynamic`: 表示是否启用动态调整 real server 节点权重，“!misc_dynamic”表示不启用，相反则表示启用。在启用这个功能后，Keepalived 的 healthchecker 进程将通过退出状态码来动态调整 real server 节点的 weight 值，如果返回状态码为 0，表示健康检查正常，real server 节点权重保持不变；如果返回状态码为 1，表示健康检查失败，那么将 real server 节点权重设置为 0；如果返回状态码为 2 ~ 255 之间任意数值，表示健康检查正常，但 real server 节点的权重将被设置为返回状态码减 2，例如返回状态码为 10，real server 节点权重将被设置为 8 (10-2)。

到这里为止，Keepalived 配置文件中常用的选项已经介绍完毕。在默认情况下，Keepalived 在启动时会查找 /etc/Keepalived/Keepalived.conf 配置文件，如果配置文件放在其他路径下，通过“Keepalived -f”参数指定配置文件的路径即可。

在配置 Keepalived.conf 时，需要特别注意配置文件的语法格式，因为 Keepalived 在启动时并不检测配置文件的正确性，即使没有配置文件，Keepalived 照样能够启动，所以一定要保证配置文件正确。

11.3 Keepalived 基础功能应用实例

作为一个高可用集群软件，Keepalived 没有 Heartbeat、RHCS 等专业的高可用集群软件功能强大，它不能实现集群资源的托管，也不能实现对集群中运行服务的监控，但是这并不妨碍 Keepalived 的易用性，它提供了 `vrrp_script`、`notify_master`、`notify_backup` 等多个功能模块，通过这些模块也可以实现对集群资源的托管以及集群服务的监控。

11.3.1 Keepalived 基础 HA 功能演示

在默认情况下，Keepalived 可以实现对系统死机、网络异常及 Keepalived 本身进行监控，也就是说当系统出现死机、网络出现故障或 Keepalived 进程异常时，Keepalived 会进行主备节点的切换。但这些还是不够的，因为集群中运行的服务也随时可能出现问題，因此，还需要对集群中运行服务的状态进行监控，当服务出现问题时也进行主备切换。Keepalived 作为一个优秀的高可用集群软件，也考虑到了这一点，它提供了一个 `vrrp_script` 模块专门用来对集群中服务资源进行监控。

1. 配置 Keepalived

下面将通过配置一套 Keepalived 集群系统来演示 Keepalived 高可用集群的实现过程。这里以操作系统 CentOS release 6.3、keepalived-1.2.12 版本为例，更具体的集群部署环境如表 11-1 所示。

表 11-1 Keepalived 高可用集群环境部署说明

主机名	主机 IP 地址	集群角色	集群服务	虚拟 IP 地址
keepalived-master	192.168.66.11	MASTER (主节点)	HTTPD	192.168.66.80
keepalived-backup	192.168.66.12	BACKUP (备用节点)	HTTPD	

通过此表可以看出，这里要部署一套基于 HTTPD 的高可用集群系统。

关于 Keepalived 的安装，11.2 节已经做过详细介绍，这里不再多说，下面给出 keepalived-master 节点的 keepalived.conf 文件的内容。

```
global_defs {
    notification_email {
        acassen@firewall.loc
        failover@firewall.loc
        sysadmin@firewall.loc
    }
    notification_email_from Alexandre.Cassen@firewall.loc
    smtp_server 192.168.200.1
    smtp_connect_timeout 30
    router_id LVS_DEVEL
}

vrrp_script check_httpd {
    script "killall -0 httpd"
    interval 2
}

vrrp_instance HA_1 {
    state MASTER
    interface eth0
    virtual_router_id 80
    priority 100
    advert_int 2
    authentication {
        auth_type PASS
        auth_pass qwaszx
    }
}
```

```

notify_master "/etc/keepalived/master.sh"
notify_backup "/etc/keepalived/backup.sh"
notify_fault "/etc/keepalived/fault.sh"

```

```

track_script {
    check_httpd
}

```

```

virtual_ipaddress {
    192.168.66.80/24 dev eth0
}

```

其中，master.sh 文件的内容为：

```

#!/bin/bash
LOGFILE=/var/log/keepalived-mysql-state.log
echo "[Master]" >> $LOGFILE
date >> $LOGFILE

```

backup.sh 文件的内容为：

```

#!/bin/bash
LOGFILE=/var/log/keepalived-mysql-state.log
echo "[Backup]" >> $LOGFILE
date >> $LOGFILE

```

fault.sh 文件的内容为：

```

#!/bin/bash
LOGFILE=/var/log/keepalived-mysql-state.log
echo "[Fault]" >> $LOGFILE
date >> $LOGFILE

```

这三个脚本的作用是监控 Keepalived 角色的切换过程，从而帮助读者理解 notify 参数的执行过程。

keepalived-backup 节点上的 keepalived.conf 配置文件内容与 keepalived-master 节点上的基本相同，需要修改的地方有两个：

- 将“state MASTER”更改为“state BACKUP”。
- 将 priority 100 更改为一个较小的值，这里改为“priority 80”。

2. Keepalived 启动过程分析

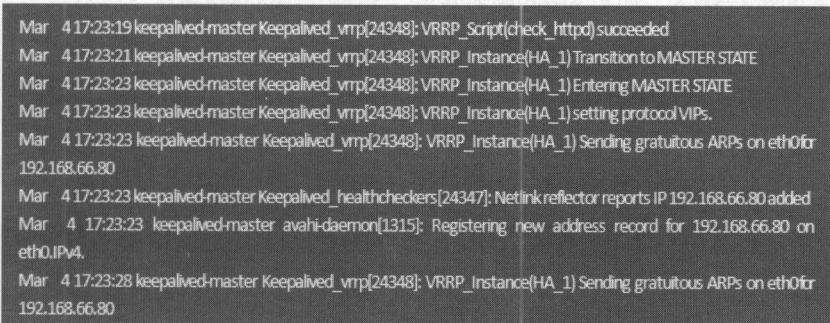
将配置好的 keepalived.conf 文件及 master.sh、backup.sh、fault.sh 三个文件一起复制到 keepalived-backup 备用节点对应的路径下，然后在两个节点启动 HTTP 服务，最后启动

Keepalived 服务。下面介绍具体的操作过程。

首先在 keepalived-master 节点启动 Keepalived 服务，执行如下操作：

```
[root@keepalived-master keepalived]# chkconfig --level 35 httpd on
[root@keepalived-master keepalived]# /etc/init.d/httpd start
[root@keepalived-master keepalived]# /etc/init.d/keepalived start
```

Keepalived 正常运行后共启动了三个进程，其中一个进程是父进程，负责监控其余两个子进程（分别是 vrrp 子进程和 healthcheckers 子进程）。然后观察 keepalived-master 上 Keepalived 的运行日志，信息如图 11-3 所示。



```
Mar 4 17:23:19 keepalived-master Keepalived_vrrp[24348]: VRRP_Script(check_httpd) succeeded
Mar 4 17:23:21 keepalived-master Keepalived_vrrp[24348]: VRRP_Instance(HA_1) Transition to MASTER STATE
Mar 4 17:23:23 keepalived-master Keepalived_vrrp[24348]: VRRP_Instance(HA_1) Entering MASTER STATE
Mar 4 17:23:23 keepalived-master Keepalived_vrrp[24348]: VRRP_Instance(HA_1) setting protocol VIPs.
Mar 4 17:23:23 keepalived-master Keepalived_vrrp[24348]: VRRP_Instance(HA_1) Sending gratuitous ARPs on eth0 for 192.168.66.80
Mar 4 17:23:23 keepalived-master Keepalived_healthcheckers[24347]: Netlink reflector reports IP 192.168.66.80 added
Mar 4 17:23:23 keepalived-master avahi-daemon[1315]: Registering new address record for 192.168.66.80 on eth0:IPv4.
Mar 4 17:23:28 keepalived-master Keepalived_vrrp[24348]: VRRP_Instance(HA_1) Sending gratuitous ARPs on eth0 for 192.168.66.80
```

图 11-3 keepalived-master 上 Keepalived 的启动日志

从日志可以看出，在 keepalived-master 主节点启动后，VRRP_Script 模块首先运行了 check_httpd 的检查，发现 httpd 服务运行正常，然后进入 MASTER 角色，如果检查 httpd 服务异常，将进入 FAULT 状态，最后将虚拟 IP 地址添加到系统中，完成 Keepalived 在主节点的启动。此时在主节点通过命令“ip add”就能查看到已经添加到系统中的虚拟 IP 地址。

再看看 /var/log/keepalived-mysql-state.log 日志文件，内容如下：

```
[root@keepalived-master keepalived]# tail -f /var/log/keepalived-mysql-state.log
[Master]

Tue Mar 4 17:23:23 CST 2014
```

通过上面给出的三个脚本的内容可知，Keepalived 在切换到 MASTER 角色后，执行了 /etc/keepalived/master.sh 这个脚本，从这里也可以看出 notify_master 的作用。

接着在备用节点 keepalived-backup 上也启动 Keepalived 服务，执行如下操作：

```
[root@keepalived-backup keepalived]# chkconfig --level 35 httpd on
[root@keepalived-backup keepalived]# /etc/init.d/httpd start
[root@keepalived-backup keepalived]# /etc/init.d/keepalived start
```

然后观察 keepalived-backup 上 Keepalived 的运行日志，信息如图 11-4 所示。

```

Mar 4 17:27:15 keepalived-backup Keepalived_healthcheckers[25912]: Opening file /etc/keepalived/keepalived.conf.
Mar 4 17:27:15 keepalived-backup Keepalived_healthcheckers[25912]: Configuration is using : 7500 Bytes
Mar 4 17:27:15 keepalived-backup Keepalived_vrrp[25913]: VRRP_Instance(HA_1) Entering BACKUP STATE
Mar 4 17:27:15 keepalived-backup Keepalived_vrrp[25913]: VRRP sockpool: [ifindex(2), proto(112), unicast(0), fd(10,11)]
Mar 4 17:27:15 keepalived-backup Keepalived_healthcheckers[25912]: Using LinkWatch kernel netlink reflector...
Mar 4 17:27:15 keepalived-backup Keepalived_vrrp[25913]: VRRP_Script(check_httpd)succeeded

```

图 11-4 keepalived-backup 上 Keepalived 启动日志

从日志输出可以看出，keepalived-backup 备用节点在启动 Keepalived 服务后，由于自身角色为 BACKUP，所以会首先进入 BACKUP 状态，接着也会运行 VRRP_Script 模块检查 httpd 服务的运行状态，如果 httpd 服务正常，将输出 succeeded。

在备用节点查看下 /var/log/keepalived-mysql-state.log 日志文件，内容如下：

```

[root@keepalived-backup keepalived]# tail -f /var/log/keepalived-mysql-state.log
[Backup]
Tue Mar 4 17:27:15 CST 2014

```

由此可知，备用节点在切换到 BACKUP 状态后，执行了 /etc/keepalived/backup.sh 这个脚本。

3. Keepalived 的故障切换过程分析

下面开始测试一下 Keepalived 的故障切换（failover）功能，首先在 keepalived-master 节点关闭 httpd 服务，然后看看 Keepalived 是如何实现故障切换的。

在 keepalived-master 节点关闭 httpd 服务后，紧接着查看 Keepalived 的运行日志，操作如图 11-5 所示。

```

[root@keepalived-master keepalived]# killall -9 httpd
[root@keepalived-master keepalived]# tail -f /var/log/messages
Mar 4 18:22:17 keepalived-master Keepalived_vrrp[24348]: VRRP_Script(check_httpd) failed
Mar 4 18:22:19 keepalived-master Keepalived_vrrp[24348]: VRRP_Instance(HA_1) Entering FAULT STATE
Mar 4 18:22:19 keepalived-master Keepalived_vrrp[24348]: VRRP_Instance(HA_1) removing protocol VIPs.
Mar 4 18:22:19 keepalived-master Keepalived_vrrp[24348]: VRRP_Instance(HA_1) Now in FAULT state
Mar 4 18:22:19 keepalived-master avahi-daemon[1315]: Withdrawing address record for 192.168.66.80 on eth0.
Mar 4 18:22:19 keepalived-master Keepalived_healthcheckers[24347]: Netlink reflector reports IP 192.168.66.80 removed

```

图 11-5 keepalived-master 节点在 httpd 服务故障后的日志状态

从日志可以看出，在关闭 keepalived-master 节点的 httpd 服务后，vrrp_script 模块很快就能检测到，然后进入 FAULT 状态，最后将虚拟 IP 地址从 eth0 上移除。

紧接着查看 keepalived-backup 节点上 Keepalived 的运行日志，信息如图 11-6 所示。


```

Mar 4 18:24:00 keepalived-backup Keepalived_vrrp[27793]: VRRP_Instance(HA_1) Transition to MASTER STATE
Mar 4 18:24:02 keepalived-backup Keepalived_vrrp[27793]: VRRP_Instance(HA_1) Entering MASTER STATE
Mar 4 18:24:02 keepalived-backup Keepalived_vrrp[27793]: VRRP_Instance(HA_1) setting protocol VIPs.
Mar 4 18:24:02 keepalived-backup Keepalived_vrrp[27793]: VRRP_Instance(HA_1) Sending gratuitous ARPs on eth0 for
192.168.66.80
Mar 4 18:24:02 keepalived-backup avahi-daemon[1207]: Registering new address record for 192.168.66.80 on eth0:IPv4.
Mar 4 18:24:02 keepalived-backup Keepalived_healthcheckers[27792]: Netlink reflector reports IP 192.168.66.80 added
Mar 4 18:24:07 keepalived-backup Keepalived_vrrp[27793]: VRRP_Instance(HA_1) Sending gratuitous ARPs on eth0 for
192.168.66.80

```

图 11-6 keepalived-backup 接管故障节点后的日志状态

从日志可以看出，在 keepalived-master 节点出现故障后，备用节点 keepalived-backup 立刻检测到，此时备用机变为 MASTER 角色，并且接管了 keepalived-master 主机的虚拟 IP 资源，最后将虚拟 IP 绑定在 eth0 设备上。

Keepalived 在发生故障时进行切换的速度是非常快的，只有几秒的时间，如果在切换过程中，持续 ping 虚拟 IP 地址，可以发现几乎没有延时等待时间。

4. 故障恢复切换分析

由于设置了集群中的主、备节点角色，因此，主节点在恢复正常后会自动再次从备用节点夺取集群资源，这是常见高可用集群系统的运行原理。下面继续演示故障恢复后 Keepalived 的切换过程。

首先在 keepalived-master 节点上启动 httpd 服务：

```
[root@keepalived-master ~]# /etc/init.d/httpd start
```

其次查看 Keepalived 的运行日志，信息如图 11-7 所示。

```

Mar 4 20:11:58 keepalived-master Keepalived_vrrp[24348]: VRRP_Script(check_httpd)succeeded
Mar 4 20:11:58 keepalived-master Keepalived_vrrp[24348]: VRRP_Instance(HA_1) prio is higher than received advert
Mar 4 20:11:58 keepalived-master Keepalived_vrrp[24348]: VRRP_Instance(HA_1) Transition to MASTER STATE
Mar 4 20:11:58 keepalived-master Keepalived_vrrp[24348]: VRRP_Instance(HA_1) Received lower prio advert, forcing
new election
Mar 4 20:12:00 keepalived-master Keepalived_vrrp[24348]: VRRP_Instance(HA_1) Entering MASTER STATE
Mar 4 20:12:00 keepalived-master Keepalived_vrrp[24348]: VRRP_Instance(HA_1) setting protocol VIPs.
Mar 4 20:12:00 keepalived-master Keepalived_vrrp[24348]: VRRP_Instance(HA_1) Sending gratuitous ARPs on eth0 for
192.168.66.80
Mar 4 20:12:00 keepalived-master Keepalived_healthcheckers[24347]: Netlink reflector reports IP 192.168.66.80 added
Mar 4 20:12:00 keepalived-master avahi-daemon[1315]: Registering new address record for 192.168.66.80 on eth0:IPv4.
Mar 4 20:12:05 keepalived-master Keepalived_vrrp[24348]: VRRP_Instance(HA_1) Sending gratuitous ARPs on eth0 for
192.168.66.80

```

图 11-7 keepalived-master 节点 httpd 服务恢复后 Keepalived 的运行日志

从日志可知，keepalived-master 节点通过 vrrp_script 模块检测到 httpd 服务已经恢复正常，然后自动切换到 MASTER 状态，同时也夺回了集群资源，将虚拟 IP 地址再次绑定在

eth0 设备上。

继续查看 keepalived-backup 节点 Keepalived 的运行日志信息，如图 11-8 所示。

```
Mar 4 20:13:51 keepalived-backup Keepalived_vrrp[27793]: VRRP_Instance(HA_1) Received higher prio advert
Mar 4 20:13:51 keepalived-backup Keepalived_vrrp[27793]: VRRP_Instance(HA_1) Entering BACKUP STATE
Mar 4 20:13:51 keepalived-backup Keepalived_vrrp[27793]: VRRP_Instance(HA_1) removing protocol VIPs.
Mar 4 20:13:51 keepalived-backup Keepalived_healthcheckers[27792]: Netlink reflector reports IP 192.168.66.80
removed
Mar 4 20:13:51 keepalived-backup avahi-daemon[1207]: Withdrawing address record for 192.168.66.80 on eth0
```

图 11-8 keepalived-backup 节点在 keepalived-master 节点故障恢复后的日志信息

从图 11-8 中可以看出，keepalived-backup 节点在发现主节点恢复正常后，释放了集群资源，重新进入了 BACKUP 状态，于是整个集群系统恢复了正常的主、备运行状态。

纵观 Keepalived 的整个运行过程和切换过程，看似合理，事实上并非如此：在一个高负载、高并发、追求稳定的业务系统中，执行一次主、备切换对业务系统影响很大，因此，不到万不得已，尽量不要进行主、备角色的切换，也就是说，在主节点发生故障后，必须切换到备用节点，而在主节点故障恢复后，不希望再次切回主节点，直到备用节点发生故障时才进行切换，这就是前面介绍过的不抢占功能，可以通过 Keepalived 的 nopreempt 选项来实现。

11.3.2 通过 vrrp_script 实现对集群资源的监控

在上节介绍 Keepalived 基础 HA 功能时用到了 vrrp_script 这个模块，此模块专门用于对集群中服务资源进行监控。与此模块一起使用的还有 track_script 模块，在此模块中可以引入监控脚本、命令组合、shell 语句等，以实现对服务、端口等多方面的监控。track_script 模块主要用来调用“vrrp_script”模块使 Keepalived 执行对集群服务资源的检测。

此外，在 vrrp_script 模块中还可以定义对服务资源检测的时间间隔、权重等参数，通过 vrrp_script 和 track_script 组合，可以实现对集群资源的监控并改变集群优先级，进而实现 Keepalived 的主、备节点切换。

下面就详细介绍下 vrrp_script 模块常见的几种监测机制，至于选择哪种监控方面，视实际应用环境而定。

1. 通过 killall 命令探测服务运行状态

这种监控集群服务的方式主要是通过 killall 命令实现的。killall 会发送一个信号到正在运行的指定命令的进程。如果没指定信号名，则发送 SIGTERM。SIGTERM 也是信号名的一种，代号为 15，它表示以正常的方式结束程序的运行。其实 killall 可用的信号名有很多，可通过“killall -l”命令显示所有信号名列表，其中每个信号名代表对进程的不同执行方式，

例如，代号为 9 的信号表示将强制中断一个程序的运行。这里要用到的信号为 0，代号为 0 的信号并不表示要关闭某个程序，而表示对程序（进程）的运行状态进行监控，如果发现进程关闭或其他异常，将返回状态码 1，反之，如果发现进程运行正常，将返回状态码 0。vrrp_script 模块正是利用了 killall 命令的这个特性，变相实现了对服务运行状态的监控。下面看一个实例：

```
vrrp_script check_mysql {
    script "killall -0 mysql"
    interval 2
}
track_script {
    check_mysql
}
```

这个例子定义了一个服务监控模块 check_mysql，其采用的监控的方式是通过“killall -0 mysql”的方式，其中“interval”选项检查的时间间隔，即 2s 执行一次检测。

在 MySQL 服务运行正常情况下，通过 killall 命令检测结果如下：

```
[root@keepalived-master ~]# killall -0 mysql
[root@keepalived-master ~]# echo $?
0
```

这里通过“echo \$?”方式显示了上个命令的返回状态码，MySQL 服务运行正常，因此返回的状态码为 0，此时 check_mysql 模块将返回服务检测正常的提示。接着将 MySQL 服务关闭，再次执行检测，结果如下：

```
[root@keepalived-master ~]# killall -0 mysql
mysql: no process killed
[root@keepalived-master ~]# echo $?
1
```

由于 MySQL 服务被关闭，因此返回的状态码为 1，此时 check_mysql 模块将返回服务检测失败的提示。然后根据 vrrp_script 模块中设定的“weight”值重新设置 Keepalived 主、备节点的优先级，进而引发主、备节点发生切换。

从这个过程可以看到，vrrp_script 模块其实并不关注监控脚本或监控命令是如何实现的，它仅仅通过监控脚本的返回状态码来识别集群服务是否正常，如果返回状态码为 0，那么就认为服务正常，如果返回状态码为 1，则认为服务故障。明白了这个原理之后，在进行自定义监控脚本的时候，只需按照这个原则来编写即可。

2. 检测端口运行状态

检测端口的运行状态也是最常见的服务监控方式，在 Keepalived 的 vrrp_script 模块中

可以通过如下方式对本机的端口进行检测：

```
vrrp_script check_httpd {
    script "</dev/tcp/127.0.0.1/80"
    interval 2
    fall 2
    rise 1
}
track_script {
    check_httpd
}
```

在这个例子中，通过“</dev/tcp/127.0.0.1/80”这样的方式定义了一个对本机 80 端口的状态检测，其中，“fall”选项表示检测到失败的最大次数，也就是说，如果请求失败两次，就认为此节点资源发生故障，将进行切换操作；“rise”表示如果请求一次成功，就认为此节点资源恢复正常。

3. 通过 shell 语句进行状态监控

在 Keepalived 的 vrrp_script 模块中甚至可以直接引用 shell 语句进行状态监控，例如下面这个示例：

```
vrrp_script chk_httpd {
    script "if [ -f /var/run/httpd/httpd.pid ]; then exit 0; else exit 1; fi"
    interval 2
    fall 1
    rise 1
}
track_script {
    chk_httpd
}
```

在这个例子中，通过一个 shell 判断语句，检测 httpd.pid 文件是否存在，如果存在，就认为状态正常，否则认为状态异常，这种监测方式对于一些简单的应用监控或者流程监控非常有用。从这里也可以得知，vrrp_script 模块支持的监控方式十分灵活。

4. 通过脚本进行服务状态监控

这是最常见的监控方式，其监控过程类似于 nagios 的执行方式，不同的是，这里只有 0、1 两种返回状态，例如下面这个示例：

```
vrrp_script chk_mysql {
    script "/etc/keepalived/check_mysql.sh"
    interval 2
```



```

}
track_script {
chk_mysqlld
}

```

其中, `check_mysqlld.sh` 的内容为:

```

#!/bin/bash
MYSQL=/usr/bin/mysql
MYSQL_HOST=localhost
MYSQL_USER=root
MYSQL_PASSWORD='xxxxxx'

$MYSQL -h $MYSQL_HOST -u $MYSQL_USER -p$MYSQL_PASSWORD -e "show status;" > /dev/
null 2>&1
if [ $? = 0 ];then
    MYSQL_STATUS=0
else
    MYSQL_STATUS=1
fi

exit $MYSQL_STATUS

```

这是一个最简单的实现 MySQL 服务状态检测的 shell 脚本, 它通过登录 MySQL 数据库后执行查询操作来检测 MySQL 运行是否正常, 如果检测正常, 将返回状态码 0, 否则返回状态码 1。其实, 很多在 nagios 下运行的脚本, 只要稍作修改, 即可在这里使用, 非常方便。

11.3.3 Keepalived 集群中 MASTER 和 BACKUP 角色选举策略

在 Keepalived 集群中, 其实并没有严格意义上的主、备节点, 虽然可以在 Keepalived 配置文件中设置 `state` 选项为 MASTER 状态, 但是这并不意味着此节点一直都是 MASTER 角色。控制节点角色的是 Keepalived 配置文件中的 `priority` 值, 但它并不控制所有节点的角色, 另一个能改变节点角色的是在 `vrrp_script` 模块中设置的 `weight` 值, 这两个选项对应的都是一个整数值, 其中 `weight` 值可以是负整数, 一个节点在集群中的角色就是通过这两个值的大小决定的。

在一个一主多备的 Keepalived 集群中, `priority` 值最大的将成为集群中的 MASTER 节点, 而其他都是 BACKUP 节点。在 MASTER 节点发生故障后, BACKUP 节点之间将进行“民主选举”, 通过对节点优先级值 `priority` 和 `weight` 的计算, 选出新的 MASTER 节点接管集群服务。

在 `vrrp_script` 模块中, 如果不设置 `weight` 选项值, 那么集群优先级的选择将由

Keepalived 配置文件中的 priority 值决定，而在需要对集群中优先级进行灵活控制时，可以通过在 vrrp_script 模块中设置 weight 值来实现。下面列举一个实例来具体说明。

假定由 A 和 B 两个节点组成的 Keepalived 集群，在 A 节点 keepalived.conf 文件中，设置 priority 值为 100，而在 B 节点 keepalived.conf 文件中，设置 priority 值为 80，并且 A、B 两个节点都使用了 vrrp_script 模块来监控 MySQL 服务，同时都设置 weight 值为 10，那么将会发生如下情况。

在两节点都启动 Keepalived 服务后，正常情况是 A 节点将成为集群中的 MASTER 节点，而 B 自动成为 BACKUP 节点，此时将 A 节点的 MySQL 服务关闭，通过查看日志发现，并没有出现 B 节点接管 A 节点的日志，B 节点仍然处于 BACKUP 状态，而 A 节点依旧处于 MASTER 状态，在这种情况下整个 HA 集群将失去意义。

下面分析产生这种情况的原因。这也就是 Keepalived 集群中主、备角色选举策略的问题。下面总结在 Keepalived 中使用 vrrp_script 模块时整个集群角色的选举算法，由于 weight 值可以是正数也可以是负数，因此，要分两种情况进行说明。

1. weight 值为正数时

在 vrrp_script 中指定的脚本如果检测成功，那么 MASTER 节点的权值将是 weight 值与 priority 值之和；如果脚本检测失败，那么 MASTER 节点的权值保持为 priority 值，因此切换策略为：

- ❑ MASTER 节点 vrrp_script 脚本检测失败时，如果 MASTER 节点 priority 值小于 BACKUP 节点 weight 值与 priority 值之和，将发生主、备切换。
- ❑ MASTER 节点 vrrp_script 脚本检测成功时，如果 MASTER 节点 weight 值与 priority 值之和大于 BACKUP 节点 weight 值与 priority 值之和，主节点依然为主节点，不发生切换。

2. weight 值为负数时

在 vrrp_script 中指定的脚本如果检测成功，那么 MASTER 节点的权值仍为 priority 值，当脚本检测失败时，MASTER 节点的权值将是 priority 值与 weight 值之差，因此切换策略为：

- ❑ MASTER 节点 vrrp_script 脚本检测失败时，如果 MASTER 节点 priority 值与 weight 值之差小于 BACKUP 节点 priority 值，将发生主、备切换。
- ❑ MASTER 节点 vrrp_script 脚本检测成功时，如果 MASTER 节点 priority 值大于 BACKUP 节点 priority 值时，主节点依然为主节点，不发生切换。

在熟悉了 Keepalived 主、备角色的选举策略后，再来分析一下前面的那个实例。由于 A、B 两个节点设置的 weight 值都为 10，因此符合选举策略的第一种，在 A 节点停止 MySQL 服

务后, A 节点的脚本检测将失败, 此时 A 节点的权值将保持为 A 节点上设置的 priority 值, 即为 100, 而 B 节点的权值将变为 weight 值与 priority 值之和, 也就是 90 (10+80), 这样就出现了 A 节点权值仍然大于 B 节点权值的情况, 因此不会发生主、备切换。

对于 weight 值的设置, 有一个简单的标准, 即 weight 值的绝对值要大于 MASTER 和 BACKUP 节点 priority 值之差。对于上面 A、B 两个节点的例子, 只要设置 weight 值大于 20 即可保证集群正常运行和切换。由此可见, 对于 weight 值的设置要非常谨慎, 如果设置不好, 主节点发生故障时将导致集群角色选举失败, 使集群陷于瘫痪状态。

12.1 高性能负载均衡软件 HAProxy 介绍

随着互联网业务的迅猛发展, 大型电商平台和门户网站对系统的可用性和可靠性要求越来越高, 高可用条件、负载均衡群成为一种热门的系统架构解决方案。在众多的负载均衡集群解决方案中, 有基于硬件的负载均衡设备, 例如 CS、Big IP 等, 也有基于软件的负载均衡产品, 例如 HAProxy、LVS、Nginx 等。在软件的负载均衡产品中, 又分为两种实现方式: 分别是基于操作系统的负载均衡实现和基于第三方应用的负载均衡实现。LVS 就是基于 Linux 操作系统实现的一种软负载均衡, 而 HAProxy 则是基于第三方应用实现的软负载均衡。本章将详细介绍 HAProxy 这种基于第三方应用实现的负载均衡技术。

12.1.1 HAProxy 简介

HAProxy 是一个开源的、高性能的、基于 TCP (第四层) 和 HTTP (第七层) 应用的负载均衡软件。借助 HAProxy 可以实现、可靠地提供基于 TCP 和 HTTP 应用的负载均衡解决方案。HAProxy 作为一个专业的负载均衡软件, 它的主要优点如下:

- 可靠性和稳定性非常好, 可以与硬件级的 FS 负载均衡设备相媲美。
- 最高可以同时维护 40 000 ~ 50 000 个并发连接, 单位时间内处理的最大请求数为 10 000 个, 最大数据吞吐量可达 10Gbps。作为软件级别的负载均衡来说, HAProxy 的性能强大可见一斑。
- 支持多于 8 种负载均衡算法, 同时也支持 session 保持。

千万级高并发负载均衡软件 HAProxy

12.1 高性能负载均衡软件 HAProxy 介绍

随着互联网业务的迅猛发展，大型电商平台和门户网站对系统的可用性和可靠性要求越来越高，高可用集群、负载均衡集群成为一种热门的系统架构解决方案。在众多的负载均衡集群解决方案中，有基于硬件的负载均衡设备，例如 F5、Big-IP 等，也有基于软件的负载均衡产品，例如 HAProxy、LVS、Nginx 等。在软件的负载均衡产品中，又分为两种实现方式，分别是基于操作系统的软负载实现和基于第三方应用的软负载实现。LVS 就是基于 Linux 操作系统实现的一种软负载均衡，而 HAProxy 就是基于第三应用实现的软负载均衡。本章将详细介绍 HAProxy 这种基于第三方应用实现的负载均衡技术。

12.1.1 HAProxy 简介

HAProxy 是一个开源的、高性能的、基于 TCP（第四层）和 HTTP（第七层）应用的负载均衡软件，借助 HAProxy 可以快速、可靠地提供基于 TCP 和 HTTP 应用的负载均衡解决方案。HAProxy 作为一个专业的负载均衡软件，它的显著优点如下：

- 可靠性和稳定性非常好，可以与硬件级的 F5 负载均衡设备相媲美。
- 最高可以同时维护 40 000 ~ 50 000 个并发连接，单位时间内处理的最大请求数为 20 000 个，最大数据处理能力可达 10Gbps。作为软件级别的负载均衡来说，HAProxy 的性能强大可见一斑。
- 支持多于 8 种负载均衡算法，同时也支持 session 保持。

- ❑ 支持虚拟主机功能，这样实现 Web 负载均衡更加灵活。
- ❑ 从 HAProxy1.3 版本后开始支持连接拒绝、全透明代理等功能，这些功能是所有其他负载均衡器所不具备的。
- ❑ HAProxy 拥有一个功能强大的服务器状态监控页面，通过此页面可以实时了解系统的运行状况。
- ❑ HAProxy 拥有功能强大的 ACL 支持，能给使用带来很大方便。

HAProxy 是借助于操作系统的技术特性来实现性能最大化的，因此，在使用 HAProxy 时，对操作系统进行性能调优是非常重要的。在业务系统方面，HAProxy 非常适用于那些并发量特别大且需要持久连接或四层和七层处理机制的 Web 系统，例如门户网站或电商网站等。另外，HAProxy 也可用于 MySQL 数据库（读操作）的负载均衡。

12.1.2 四层和七层负载均衡的区别

在 12.1 节中提到了 HAProxy 是一个四层和七层负载均衡器。下面简单了解下四层和七层的概念与区别。

所谓的四层就是 ISO 参考模型中的第四层。四层负载均衡器也称为四层交换机，它主要是通过分析 IP 层及 TCP/UDP 层的流量实现的基于“IP+ 端口”的负载均衡。常见的基于四层的负载均衡器有 LVS、F5 等。

以常见的 TCP 应用为例，负载均衡器在接收到第一个来自客户端的 SYN 请求时，会通过设定的负载均衡算法选择一台最佳的后端服务器，同时将报文中目标 IP 地址修改为后端服务器 IP，然后直接转发给该后端服务器，这样一个负载均衡请求就完成了。从这个过程来看，一个 TCP 连接是客户端和服务器直接建立的，而负载均衡器只不过完成了一个类似路由器的转发动作。在某些负载均衡策略中，为保证后端服务器返回的报文可以正确传递给负载均衡器，在转发报文的同时可能还会对报文原来的源地址进行修改。整个过程如图 12-1 所示。

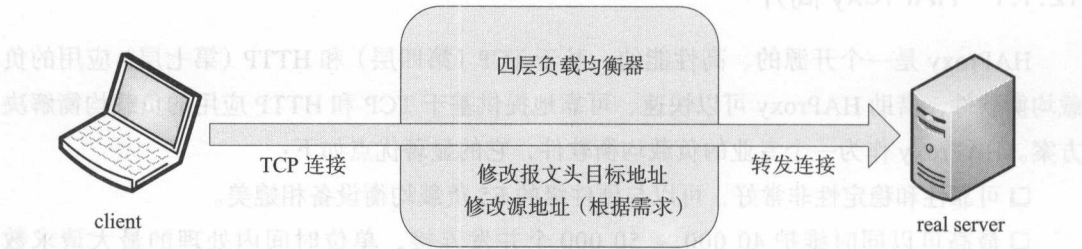


图 12-1 四层负载均衡器转发原理

同理，七层负载均衡器也称为七层交换机，位于 ISO 的最高层，即应用层，此时负载均衡器支持多种应用协议，常见的有 HTTP、FTP、SMTP 等。七层负载均衡器可以根据报

文内容，再配合负载均衡算法来选择后端服务器，因此也称为“内容交换器”。比如，对于 Web 服务器的负载均衡，七层负载均衡器不但可以根据“IP+端口”的方式进行负载分流，还可以根据网站的 URL、访问域名、浏览器类别、语言等决定负载均衡的策略。例如，有两台 Web 服务器分别对应中英文两个网站，两个域名分别是 A、B，要实现访问 A 域名时进入中文网站，访问 B 域名时进入英文网站，这在四层负载均衡器中几乎是无法实现的，而七层负载均衡器可以根据客户端访问域名的不同选择对应的网页进行负载均衡处理。常见的七层负载均衡器有 HAProxy、Nginx 等。

这里仍以常见的 TCP 应用为例，由于负载均衡器要获取到报文的内容，因此只能先代替后端服务器和客户端建立连接，接着，才能收到客户端发送过来的报文内容，然后再根据该报文中特定字段加上负载均衡器中设置的负载均衡器算法来决定最终选择的内部服务器。纵观整个过程，七层负载均衡器在这种情况下类似于一个代理服务器。整个过程如图 12-2 所示。

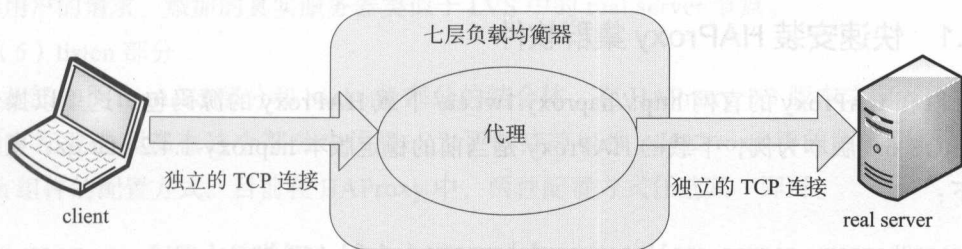


图 12-2 七层负载均衡器代理实现原理

对比四层负载均衡器和七层负载均衡器运行的整个过程，可以看出，在七层负载均衡模式下，负载均衡器与客户端及后端的服务器会分别建立一次 TCP 连接，而在四层负载均衡模式下，仅建立一次 TCP 连接。由此可知，七层负载均衡对负载均衡设备的要求更高，而七层负载均衡的处理能力也必然低于四层模式的负载均衡。

12.1.3 HAProxy 与 LVS 的异同

通过上面两小节介绍，读者应该基本清楚了 HAProxy 负载均衡与 LVS 负载均衡的优缺点和异同了。下面就这两种负载均衡软件的异同做一个简单总结。

1) 两者都是软件负载均衡产品，但是 LVS 是基于 Linux 操作系统实现的一种软负载均衡，而 HAProxy 是基于第三应用实现的软负载均衡。

2) LVS 是基于四层的 IP 负载均衡技术，而 HAProxy 是基于四层和七层技术、可提供 TCP 和 HTTP 应用的负载均衡综合解决方案。

3) LVS 工作在 ISO 模型的第四层，因此其状态监测功能单一，而 HAProxy 在状态监

测方面功能强大，可支持端口、URL、脚本等多种状态检测方式。

4) 虽然 HAProxy 功能强大，但是它的整体处理性能低于四层负载均衡模式的 LVS，而 LVS 拥有接近硬件设备的网络吞吐和连接负载能力。

综上所述，HAProxy 和 LVS 各有优缺点，没有好坏之分，要选择哪个作为负载均衡器，要以实际的应用环境来决定。

12.2 HAProxy 基础配置与应用实例

HAProxy 的安装非常简单，但是在配置方面稍微复杂了些。虽然官方给出的配置文档多达百页，但是 HAProxy 的配置并非这么复杂，因为 HAProxy 常用的配置选项是非常少的。只要掌握了常用的配置选项，基本就能玩转 HAProxy 了，因此在下面的介绍中主要讲解 HAProxy 中常用的选项。

12.2.1 快速安装 HAProxy 集群软件

可以在 HAProxy 的官网 <http://haproxy.1wt.eu/> 下载 HAProxy 的源码包，这里以操作系统 CentOS 6.3 版本为例，下载的 HAProxy 是当前的稳定版本 haproxy-1.4.24.tar.gz，安装过程如下：

```
[root@haproxy-server app]# tar zxvf haproxy-1.4.24.tar.gz
[root@haproxy-server app]# cd haproxy-1.4.24
[root@haproxy-server haproxy-1.4.24]# make TARGET=linux26 PREFIX=/usr/local/haproxy
[root@haproxy-server haproxy-1.4.24]# make install PREFIX=/usr/local/haproxy
# 将 HAProxy 安装到 /usr/local/haproxy 下
[root@haproxy-server haproxy-1.4.24]# mkdir /usr/local/haproxy/conf
# HAProxy 默认不创建配置文件目录，这里是创建 HAProxy 配置文件目录
[root@haproxy-server haproxy-1.4.24]# cp examples/haproxy.cfg /usr/local/haproxy/conf
# HAProxy 安装完成后，默认安装目录中没有配置文件，这里是将源码包里面的示例配置文件复制到
# 配置文件目录
```

这样，HAProxy 就安装完成了。

12.2.2 HAProxy 基础配置文件详解

1. 配置文件概述

根据功能和用途，HAProxy 配置文件主要由 5 个部分组成，但有些部分并不是必需的，可以根据需要选择相应的部分进行配置。

(1) global 部分

用来设定全局配置参数，属于进程级的配置，通常和操作系统配置有关。

(2) defaults 部分

默认参数的配置部分。在此部分设置的参数值，默认会自动引用到下面的 frontend、backend 和 listen 部分中，因此，如果某些参数属于公用的配置，只需在 defaults 部分添加一次即可。而如果在 frontend、backend 和 listen 部分中也配置了与 defaults 部分一样的参数，那么 defaults 部分参数对应的值自动被覆盖。

(3) frontend 部分

此部分用于设置接收用户请求的前端虚拟节点。frontend 是在 HAProxy 1.3 版本之后才引入的一个组件，同时引入的还有 backend 组件。通过引入这些组件，在很大程度上简化了 HAProxy 配置文件的复杂性。frontend 可以根据 ACL 规则直接指定要使用的后端 backend。

(4) backend 部分

此部分用于设置集群后端服务集群的配置，也就是用来添加一组真实服务器，以处理前端用户的请求。添加的真实服务器类似于 LVS 中的 real server 节点。

(5) listen 部分

此部分是 frontend 部分和 backend 部分的结合体。在 HAProxy 1.3 版本之前，HAProxy 的所有配置选项都在这个部分中设置。为了保持兼容性，HAProxy 新的版本仍然保留了 listen 组件的配置方式。目前在 HAProxy 中，两种配置方式任选其一即可。

2. HAProxy 配置文件详解

根据上面介绍的 5 个部分对 HAProxy 的配置文件进行讲解。

(1) global 部分

配置示例如下：

```
global
    log 127.0.0.1 local0 info
    maxconn 4096
    user nobody
    group nobody
    daemon
    nbproc 1
    pidfile /usr/local/haproxy/logs/haproxy.pid
```

上述代码中每个选项的含义如下。

□ log：全局的日志配置，local0 是日志设备，info 表示日志级别。其中日志级别有 err、warning、info、debug4 种可选。这个配置表示使用 127.0.0.1 上的 rsyslog 服务中的 local0 日志设备，记录日志等级为 info。

- ❑ **maxconn**：设定每个 HAProxy 进程可接受的最大并发连接数，此选项等同于 Linux 命令行选项 “ulimit -n”。
- ❑ **user/ group**：设置运行 HAProxy 进程的用户和组，也可使用用户和组的 uid 和 gid 值来替代。
- ❑ **daemon**：设置 HAProxy 进程进入后台运行。这是推荐的运行模式。
- ❑ **nbproc**：设置 HAProxy 启动时可创建的进程数，此参数要求将 HAProxy 运行模式设置为 daemon，默认只启动一个进程。根据使用经验，该值的设置应该小于服务器的 CPU 核数。创建多个进程，能够减少每个进程的任务队列，但是过多的进程可能会导致进程崩溃。
- ❑ **pidfile**：指定 HAProxy 进程的 pid 文件。启动进程的用户必须有访问此文件的权限。

(2) defaults 部分

配置示例如下：

```
defaults
    mode http
    retries 3
    timeout connect 10s
    timeout client 20s
    timeout server 30s
    timeout check 5s
```

上述代码中每个选项的含义如下。

- ❑ **mode**：设置 HAProxy 实例默认的运行模式，有 tcp、http、health 三个可选值。
 - ❑ **tcp 模式**：在此模式下，客户端和服务器端之间将建立一个全双工的连接，不会对七层报文做任何类型的检查，默认为 tcp 模式，经常用于 SSL、SSH、SMTP 等应用。
 - ❑ **http 模式**：在此模式下，客户端请求在转发至后端服务器之前将会被深度分析，所有不与 RFC 格式兼容的请求都会被拒绝。
 - ❑ **health 模式**：目前此模式基本已经废弃，不再多说。
- ❑ **retries**：设置连接后端服务器的失败重试次数，如果连接失败的次数超过这里设置的值，HAProxy 会将对应的后端服务器标记为不可用。此参数也可在后面部分进行设置。
- ❑ **timeout connect**：设置成功连接到一台服务器的最长等待时间，默认单位是毫秒，但也可以使用其他的时间单位后缀。
- ❑ **timeout client**：设置连接客户端发送数据时最长等待时间，默认单位是毫秒，也可以使用其他的时间单位后缀。
- ❑ **timeout server**：设置服务器端回应客户端数据发送的最长等待时间，默认单位是毫秒，也可以使用其他的时间单位后缀。

□ **timeout check**: 设置对后端服务器的检测超时时间, 默认单位是毫秒, 也可以使用其他的时间单位后缀。

(3) frontend 部分

这是 HAProxy 配置文件的第三部分——frontend 部分的配置, 配置示例如下:

```
frontend www
    bind *:80
    mode http
    option httplog
    option forwardfor
    option httpclose
    log global
    default_backend httpool
```

这部分通过 frontend 关键字定义了一个名为“www”的前端虚拟节点, 上述代码中每个选项的含义如下。

□ **bind**: 此选项只能在 frontend 和 listen 部分进行定义, 用于定义一个或几个监听的套接字。bind 的使用格式为:

```
bind [<address>:<port_range>] interface <interface>
```

其中, address 为可选选项, 其可以为主机名或 IP 地址, 如果将其设置为“*”或“0.0.0.0”, 将监听当前系统的所有 IPv4 地址。port_range 可以是一个特定的 TCP 端口, 也可是一个端口范围, 小于 1024 的端口需要有特定权限的用户才能使用。interface 为可选选项, 用来指定网络接口的名称, 只能在 Linux 系统上使用。

□ **option httplog**: 在默认情况下, HAProxy 日志是不记录 HTTP 请求的, 这样很不方便 HAProxy 问题的排查与监控。通过此选项可以启用日志记录 HTTP 请求。

□ **option forwardfor**: 如果后端服务器需要获得客户端的真实 IP, 就需要配置此参数。由于 HAProxy 工作于反向代理模式, 因此发往后端真实服务器的请求中的客户端 IP 均为 HAProxy 主机的 IP, 而非真正访问客户端的地址, 这就导致真实服务器端无法记录客户端真正请求来源的 IP, 而 X-Forwarded-For 则可用于解决此问题。通过使用 forwardfor 选项, HAProxy 就可以向每个发往后端真实服务器的请求添加 X-Forwarded-For 记录, 这样后端真实服务器日志可以通过“X-Forwarded-For”信息来记录客户端来源 IP。

□ **option httpclose**: 此选项表示在客户端和服务端完成一次连接请求后, HAProxy 将主动关闭此 TCP 连接。这是对性能非常有帮助的一个参数。

□ **log global**: 表示使用全局的日志配置, 这里的 global 表示引用在 HAProxy 配置文件 global 部分中定义的 log 选项配置格式。

❑ **default_backend**：指定默认的后端服务器池，也就是指定一组后端真实服务器，而这些真实服务器组将在 **backend** 段进行定义。这里的 **htmpool** 就是一个后端服务器组。

(4) backend 部分

接着介绍的是 HAProxy 配置文件的第四部分——**backend** 部分的配置，配置示例如下：

```
backend htmpool
    mode        http
    option       redispatch
    option       abortonclose
    balance      roundrobin
    cookie       SERVERID
    option       httpchk GET /index.php
    server web1 10.200.34.181:80 cookie server1 weight 6 check inter 2000
    rise 2 fall 3
    server web2 10.200.34.182:8080 cookie server2 weight 6 check inter 2000
    rise 2 fall 3
```

这个部分通过 **backend** 关键字定义了一个名为 **htmpool** 的后端真实服务器组。上述代码中每个选项的含义如下。

- ❑ **option redispatch**：此参数用于 **cookie** 保持的环境中。在默认情况下，HAProxy 会将其请求的后端服务器的 **serverID** 插入 **cookie** 中，以保证会话的 **session** 持久性。而如果后端的服务器出现故障，客户端的 **cookie** 是不会刷新的，这就会出现问題。此时，如果设置此参数，就会将客户的请求强制定向到另外一台健康的后端服务器上，以保证服务正常。
- ❑ **option abortonclose**：如果设置了此参数，可以在服务器负载很高的情况下，自动结束当前队列中处理时间比较长的连接。
- ❑ **balance**：此关键字用来定义负载均衡算法。目前 HAProxy 支持多种负载均衡算法，常用的有如下几种：
 - ❑ **roundrobin**：是基于权重进行轮叫调度的算法，在服务器的性能分布比较均匀时，这是一种最公平、最合理的算法。此算法使用频繁。
 - ❑ **static-rr**：也是基于权重进行轮叫的调度算法，不过此算法为静态方法，在运行时调整其服务器权重不会生效。
 - ❑ **source**：是基于请求源 IP 的算法。此算法先对请求的源 IP 进行 **hash** 运算，然后将结果与后端服务器的权重总数相除后转发至某台匹配的后端服务器。这种方式可以使同一个客户端 IP 的请求始终被转发到某特定的后端服务器。
 - ❑ **leastconn**：此算法会将新的连接请求转发到具有最少连接数目的后端服务器。在

会话时间较长的场景中推荐使用此算法，例如数据库负载均衡等。此算法不适合会话较短的环境中，例如基于 HTTP 的应用。

❑ **uri**：此算法会对部分或整个 URI 进行 hash 运算，再经过与服务器的总权重相除，最后转发到某台匹配的后端服务器上。

❑ **uri_param**：此算法会根据 URL 路径中的参数进行转发，这样可保证在后端真实服务器数量不变时，同一个用户的请求始终分发到同一台机器上。

❑ **hdr(<name>)**：此算法根据 http 头进行转发，如果指定的 http 头名称不存在，则使用 roundrobin 算法进行策略转发。

❑ **cookie**：表示允许向 cookie 插入 SERVERID，每台服务器的 SERVERID 可在下面的 server 关键字中使用 cookie 关键字定义。

❑ **option httpchk**：此选项表示启用 HTTP 的服务状态检测功能。HAProxy 作为一个专业的负载均衡器，它支持对 backend 部分指定的后端服务节点的健康检查，以保证在后端 backend 中某个节点不能服务时，把从 frontend 端进来的客户端请求分配至 backend 中其他健康节点上，从而保证整体服务的可用性。option httpchk 的用法如下：

```
option httpchk <method> <uri> <version>
```

其中，各个参数的含义如下：

❑ **method**：表示 HTTP 请求的方式，常用的有 OPTIONS、GET、HEAD 几种方式。一般的健康检查可以采用 HEAD 方式进行，而不是采用 GET 方式，这是因为 HEAD 方式没有数据返回，仅检查 Response 的 HEAD 是不是状态码 200。因此，相对于 GET，HEAD 方式更快、更简单。

❑ **uri**：表示要检测的 URL 地址，通过执行此 URL，可以获取后端服务器的运行状态。在正常情况下将返回状态码 200，返回其他状态码均为异常状态。

❑ **version**：指定心跳检测时的 HTTP 的版本号。

❑ **server**：这个关键字用来定义多台后端真实服务器，不能用于 defaults 和 frontend 部分。使用格式为：

```
server <name> <address>[:port] [param*]
```

其中，每个参数含义如下：

❑ **<name>**：为后端真实服务器指定一个内部名称，随便定义一个即可。

❑ **<address>**：后端真实服务器的 IP 地址或主机名。

❑ **<port>**：指定连接请求发往真实服务器时的目标端口。在未设定时，将使用客户端请求时的同一端口。

❑ **[param*]**：为后端服务器设定的一系列参数，可用参数非常多，这里仅介绍常用

的一些参数：

- **check**：表示启用对此后端服务器执行健康状态检查。
- **inter**：设置健康状态检查的时间间隔，单位为毫秒。
- **rise**：设置从故障状态转换至正常状态需要成功检查的次数，例如，“rise 2”表示 2 次检查正确就认为此服务器可用。
- **fall**：设置后端服务器从正常状态转换为不可用状态需要检查的次数，例如，“fall 3”表示 3 次检查失败就认为此服务器不可用。
- **cookie**：为指定的后端服务器设定 cookie 值，此处指定的值将在请求入站时被检查，第一次为此值挑选的后端服务器将在后续的请求中一直被选中，其目的在于实现持久连接的功能。上面的“cookie server1”表示 web1 的 serverid 为 server1。同理，“cookie server2”表示 web2 的 serverid 为 server2。
- **weight**：设置后端真实服务器的权重，默认为 1，最大值为 256。设置为 0 表示不参与负载均衡。
- **backup**：设置后端真实服务器的备份服务器，仅仅在后端所有真实服务器均不可用的情况下才启用。

(5) listen 部分

HAProxy 配置文件的第五部分——listen 部分的配置，配置示例如下：

```
listen admin_stats
    bind 0.0.0.0:9188
    mode http
    log 127.0.0.1 local0 err
    stats refresh 30s
    stats uri /haproxy-status
    stats realm welcome login\ Haproxy
    stats auth admin:admin~!@
    stats hide-version
    stats admin if TRUE
```

这个部分通过 listen 关键字定义了一个名为“admin_stats”的实例，其实就是定义了一个 HAProxy 的监控页面，每个选项的含义如下：

- ❑ **stats refresh**：设置 HAProxy 监控统计页面自动刷新的时间。
- ❑ **stats uri**：设置 HAProxy 监控统计页面的 URL 路径，可随意指定。例如，指定“stats uri /haproxy-status”，就可以通过 <http://IP:9188/haproxy-status> 查看。
- ❑ **stats realm**：设置登录 HAProxy 统计页面时密码框上的文本提示信息。
- ❑ **stats auth**：设置登录 HAProxy 统计页面的用户名和密码。用户名和密码通过冒号分割。可为监控页面设置多个用户名和密码，每行一个。

- ❑ stats hide-version: 用来隐藏统计页面上 HAProxy 的版本信息。
- ❑ stats admin if TRUE: 通过设置此选项, 可以在监控页面上手工启用或禁用后端真实服务器, 仅在 haproxy1.4.9 以后版本有效。

至此, 完整的一个 HAProxy 配置文件介绍完毕了。当然, 这里介绍的仅仅是最常用的一些配置参数, 要深入了解 HAProxy 的功能, 可参阅官方文档。

12.2.3 HAProxy 的日志配置策略

默认情况下, HAProxy 为了节省读写 I/O 所消耗的性能, 没有自动配置日志输出功能, 但是为了维护和调试方便, 日志的输出还是很有必要的, 下面就简单介绍下如何配置 HAProxy 的日志输出功能。

由于这里使用的环境为 CentOS6.3 系统版本, 因此系统默认的日志管理方式不再是 syslog, 而是 rsyslog 管理系统日志, rsyslog 可以实现 UDP 日志接收、将日志写入文件、将日志写入数据库等功能。那么首先检测下 rsyslog 软件包是否已经安装到系统中, 操作过程如下:

```
[root@haproxy-server ~]# rpm -q rsyslog
rsyslog-5.8.10-8.el6.x86_64
```

如果已经安装了 rsyslog 软件包, 接着需要修改 rsyslog 的配置文件, 在 /etc/rsyslog.d/ 目录下创建 haproxy.conf 文件, 内容如下:

```
$ModLoad imudp
$UDPServerRun 514
local3.* /usr/local/haproxy/logs/haproxy.log
local0.* /usr/local/haproxy/logs/haproxy.log
```

这里定义了两种日志类型, 并指定了日志的输出路径, 其中:

第一行的 “imudp” 是模块名, 支持 UDP 协议。

第二行表示允许 514 端口接收使用 UDP 和 TCP 协议转发过来的日志, 而 rsyslog 在默认情况下, 正是在 514 端口监听 UDP。其实也可以将上面 haproxy.conf 文件的内容直接写到 /etc/rsyslog.conf 文件中。

然后, 还需要修改 /etc/sysconfig/rsyslog 文件, 修改为如下内容:

```
SYSDLOGD_OPTIONS="-c 2 -r -m 0"
```

其中, “-r” 表示接收远程日志。

最后重启 rsyslog 服务即可完成配置:

```
[root@haproxy-server ~]# service rsyslog restart
```

要实现将 HAProxy 日志写入指定的文件中，还需要在 haproxy.cfg 中配置对应的日志选项，这部分内容已经在上节中进行了详细介绍，这里不再说明。

12.2.4 通过 HAProxy 的 ACL 规则实现智能负载均衡

由于 HAProxy 可以工作在七层模型下，因此，要实现 HAProxy 的强大功能，一定要使用强大灵活的 ACL 规则，通过 ACL 规则可以实现基于 HAProxy 的智能负载均衡系统。HAProxy 通过 ACL 规则完成两种主要的功能，分别是：

1) 通过设置的 ACL 规则检查客户端请求是否合法。如果符合 ACL 规则要求，那么将放行；如果不符合规则，则直接中断请求。

2) 符合 ACL 规则要求的请求将被提交到后端的 backend 服务器集群，进而实现基于 ACL 规则的负载均衡。

HAProxy 中的 ACL 规则经常使用在 frontend 段中，使用方法如下：

```
acl 自定义的 acl 名称 acl 方法 -i [ 匹配的路径或文件 ]
```

其中：

□ acl：是一个关键字，表示定义 ACL 规则的开始。后面需要跟上自定义的 ACL 名称。

□ acl 方法：这个字段用来定义实现 ACL 的方法，HAProxy 定义了很多 ACL 方法，经常使用的方法有 `hdr_reg(host)`、`hdr_dom(host)`、`hdr_beg(host)`、`url_sub`、`url_dir`、`path_beg`、`path_end` 等。

□ -i：表示不区分大小写，后面需要跟上匹配的路径或文件或正则表达式。

与 ACL 规则一起使用的 HAProxy 参数还有 `use_backend`，`use_backend` 后面需要跟上一个 backend 实例名，表示在满足 ACL 规则后去请求哪个 backend 实例，与 `use_backend` 对应的还有 `default_backend` 参数，它表示在没有满足 ACL 条件的时候默认使用哪个后端 backend。

下面列举几个常见的 ACL 规则例子：

```
acl www_policy      hdr_reg(host)  -i      ^(www.z.cn|z.cn)
acl bbs_policy      hdr_dom(host)  -i      bbs.z.cn
acl url_policy      url_sub        -i      buy_sid=

use_backend          server_www     if      www_policy
use_backend          server_app     if      url_policy
use_backend          server_bbs     if      bbs_policy
default_backend      server_cache
```

这里仅仅列出了 HAProxy 配置文件中 ACL 规则的配置部分，其他选项并未列出。

这些例子定义了 `www_policy`、`bbs_policy`、`url_policy` 三个 ACL 规则，第一条规则表示如果客户端以 `www.z.cn` 或 `z.cn` 开头的域名发送请求时，则此规则返回 `true`，同理第二条

规则表示如果客户端通过 bbs.z.cn 域名发送请求时, 则此规则返回 true, 而第三条规则表示如果客户端在请求的 URL 中包含 “buy_sid=” 字符串时, 则此规则返回 true。

第四、第五、第六条规则定义了当 www_policy、bbs_policy、url_policy 三个 ACL 规则返回 true 时要调度到哪个后端 backend, 例如, 当用户的请求满足 www_policy 规则时, 那么 HAProxy 会将用户的请求直接发往名为 server_www 的后端 backend, 其他以此类推。而当用户的请求不满足任何一个 ACL 规则时, HAProxy 就会把请求发往由 default_backend 选项指定的 server_cache 这个后端 backend。

再看下面这个例子:

```
acl url_static      path_end      .gif .png .jpg .css .js
acl host_www        hdr_beg(host) -i    www
acl host_static      hdr_beg(host) -i    img. video. download. ftp.

use_backend static    if host_static || host_www url_static
use_backend www        if host_www
default_backend        server_cache
```

与上面的例子类似, 本例中也定义了 url_static、host_www 和 host_static 三个 ACL 规则, 其中, 第一条规则通过 path_end 参数定义了如果客户端在请求的 URL 中以 .gif、.png、.jpg、.css 或 .js 结尾时返回 true, 第二条规则通过 hdr_beg(host) 参数定义了如果客户端以 www 开头的域名发送请求时则返回 true, 同理, 第三条规则也是通过 hdr_beg(host) 参数定义了如果客户端以 img、video、download 或 ftp 开头的域名发送请求时则返回 true。

第四、第五条规则定义了当满足 ACL 规则后要调度到哪个后端 backend, 例如, 当用户的请求同时满足 host_static 规则与 url_static 规则, 或同时满足 host_www 和 url_static 规则时, 那么会将用户请求直接发往名为 static 的后端 backend, 如果用户请求满足 host_www 规则, 那么请求将被调度到名为 www 的后端 backend, 如果不满足所有规则, 那么将用户请求默认调度到名为 server_cache 的这个后端 backend。

12.3 基于虚拟主机的 HAProxy 负载均衡系统配置实例

前面的两节详细介绍了 HAProxy 的安装、配置以及 ACL 规则的使用, 要熟练掌握 HAProxy 的使用, 必须进行实战性操作, 那么本节将通过一个具体的实例详细讲解和演示 HAProxy 下虚拟主机的实现过程以及 HAProxy 是如何实现负载均衡和故障转移的。

12.3.1 通过 HAProxy 的 ACL 规则配置虚拟主机

下面将通过 HAProxy 的 ACL 功能配置一套基于虚拟主机的负载均衡系统, 这里的操作

系统环境为 CentOS release 6.3, HAProxy 版本为 haproxy-1.4.24, 要实现的功能如图 12-3 所示。

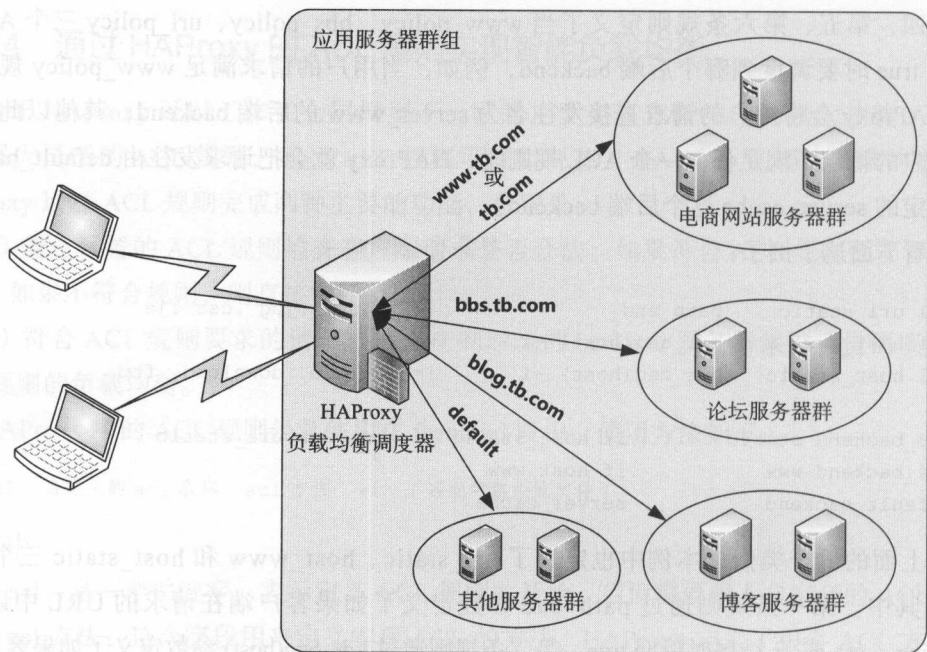


图 12-3 基于虚拟主机的 HAProxy 应用实例

本实例有一个电商网站服务器群、一个论坛服务器群、一个博客服务器群和默认服务器群，4 个服务器群都由多台服务器组成，而 4 个服务器群又组成了一个应用服务器群组，在每个服务器群的前端有一个基于 HAProxy 的负载均衡调度器，整个应用架构要实现的功能为：当客户端通过域名 `www.tb.com` 或 `tb.com` 访问时，HAProxy 将请求提交到电商网站服务器群，进而实现电商网站的负载均衡；当客户端通过域名 `bbs.tb.com` 访问时就将请求调度到论坛服务器群，实现论坛的负载均衡；当客户端通过 `blog.tb.com` 访问时则将请求调度到博客服务器群中，实现博客的负载均衡；如果客户端通过除上面三种方式外的任意方式请求服务时，就将请求调度到缺省服务器群。

要实现上述功能，如果使用四层的 LVS 负载均衡器，则需要一个代理服务器配合 LVS 负载均衡器才能实现，而通过 HAProxy 实现时，仅需要一个 HAProxy 负载调度器再结合 ACL 规则即可轻松实现。

1. 配置 HAProxy

HAProxy 的安装非常简单，前面已经做过介绍，这里直接进入 HAProxy 的配置过程，配置好的文件内容如下：

```

global
    log 127.0.0.1 local0 info
    maxconn 4096
    user nobody
    group nobody

    daemon
    nbproc 1
    pidfile /usr/local/haproxy/logs/haproxy.pid

defaults
    mode http
    retries 3
    timeout connect 5s
    timeout client 30s
    timeout server 30s
    timeout check 2s

listen admin_stats
    bind 0.0.0.0:19088
    mode http
    log 127.0.0.1 local0 err
    stats refresh 30s
    stats uri /haproxy-status
    stats realm welcome login\ Haproxy
    stats auth admin:xxxxxx
    stats auth adminl:xxxxxx
    stats hide-version
    stats admin if TRUE

frontend www
    bind *:80
    mode http
    option httplog
    option forwardfor
    log global

    acl host_www hdr_reg(host) -i ^(www.tb.com|tb.com)
    acl host_bbs hdr_dom(host) -i bbs.tb.com
    acl host_blog hdr_beg(host) -i blog.

    use_backend server_www if host_www
    use_backend server_bbs if host_bbs
    use_backend server_blog if host_blog
    default_backend server_default

backend server_default
    mode http
    option redispatch

```

```

    option abortonclose
    balance roundrobin
    cookie SERVERID
option httpchk GET /check_status.html
server default1 192.168.88.90:8000 cookie default1 weight 3 check inter
2000 rise 2 fall 3
server default2 192.168.88.91:8000 cookie default2 weight 3 check inter
2000 rise 2 fall 3

backend server_www
    mode http
    option redispatch
    option abortonclose
    balance source
    cookie SERVERID
    option httpchk GET /check_status.jsp
server www1 192.168.88.80:80 cookie www1 weight 6 check inter 2000 rise 2 fall 3
server www2 192.168.88.81:80 cookie www2 weight 6 check inter 2000 rise
2 fall 3
server www3 192.168.88.82:80 cookie www3 weight 6 check inter 2000 rise
2 fall 3

backend server_bbs
    mode http
    option redispatch
    option abortonclose
    balance source
    cookie SERVERID
    option httpchk GET /check_status.php
server bbs1 192.168.88.83:8080 cookie bbs1 weight 8 check inter 2000
rise 2 fall 3
server bbs2 192.168.88.84:8090 cookie bbs2 weight 8 check inter 2000
rise 2 fall 3

backend server_blog
    mode http
    option redispatch
    option abortonclose
    balance roundrobin
    cookie SERVERID
    option httpchk GET /check_blog.php
server blog1 192.168.88.85:80 cookie blog1 weight 5 check inter 2000
rise 2 fall 3
server blog2 192.168.88.86:80 cookie blog2 weight 5 check inter 2000
rise 2 fall 3

```

关于 HAProxy 配置文件中每个选项的含义，前面已经做过详细介绍，这里重点看一下 frontend 部分中关于 ACL 配置部分的内容，这个是实现虚拟主机的核心配置部分。另外，

这个配置文件定义了 `server_www`、`server_bbs`、`server_blog`、`server_default` 4 个 backend，分别对应上面的 4 个服务器群，对于 `server_www` 群和 `server_bbs` 群，采用了基于请求源 IP 的负载均衡算法，其他两个群均采用基于权重进行轮叫调度的算法。这也是根据 Web 应用的特点而定的。每个 backend 中都定义了 `httpchk` 的检测方式，因此要保证这里指定的 URL 页面是可访问到的。

为了验证负载均衡的功能，这里需要将后端真实服务器做一个访问标记，这个架构一共加入了 9 台后端真实服务器，共分为四组，这里将 `server_www` 的三台后端服务器默认的 Web 页面设置如下：

```
[root@www1 app]# echo "This is www1 192.168.88.80" >/var/www/html/index.html
[root@www2 app]# echo "This is www2 192.168.88.81" >/var/www/html/index.html
[root@www3 app]# echo "This is www3 192.168.88.82" >/var/www/html/index.html
```

同理，将 `server_bbs` 的两台后端服务器默认的 Web 页面设置如下：

```
[root@bbs1 app]# echo "This is bbs1 192.168.88.83" >/var/www/html/index.html
[root@bbs2 app]# echo "This is bbs2 192.168.88.84" >/var/www/html/index.html
```

接着，将 `server_blog` 的两台后端服务器默认的 Web 页面设置如下：

```
[root@blog1 app]# echo "This is blog1 192.168.88.85" >/var/www/html/index.html
[root@blog2 app]# echo "This is blog2 192.168.88.86" >/var/www/html/index.html
```

最后，将 `server_default` 的两台后端服务器默认的 Web 页面设置如下：

```
[root@default1 app]# echo "This is default1 192.168.88.90" >/var/www/html/index.html
[root@default2 app]# echo "This is default2 192.168.88.91" >/var/www/html/index.html
```

这样就为接下来的测试做好了准备。

2. 启动 HAProxy

HAProxy 安装完成后，会在安装根目录的 `sbin` 目录下生成一个可执行的二进制文件 `haproxy`，对 HAProxy 的启动、关闭、重启等维护操作都是通过这个二进制文件实现的，执行“`haproxy -h`”命令即可得到此文件的用法。

```
haproxy [-f < 配置文件 >] [-v<VD>] [-n 最大并发连接总数] [-N 默认的连接数]
```

`haproxy` 常用的参数以及含义如表 12-1 所示。

介绍完 HAProxy 常用的参数后，下面开始启动 HAProxy，操作如下：

```
[root@haproxy-server haproxy]# /usr/local/haproxy/sbin/haproxy -f \
> /usr/local/haproxy/conf/haproxy.cfg
```


表 12-1 haproxy 常用参数及含义

参数	含义
-v	显示当前版本信息，“-vv”显示已知的创建选项
-d	表示让进程运行在 debug 模式，“-db”表示禁用后台模式，让程序在前台运行
-D	让程序以 daemon 模式启动，此选项也可以在 HAProxy 配置文件中设置
-q	表示安静模式，程序运行不输出任何信息
-c	对 HAProxy 配置文件进行语法检查。此参数非常有用。如果配置文件错误，会输出对应的错误位置 and 错误信息
-n	设置最大并发连接总数
-m	限制可用的内存大小，以 MB 为单位
-N	设置默认的连接数
-p	设置 HAProxy 的 PID 文件路径
-de	不使用 epoll 模型
-ds	不使用 speculative epoll
-dp	不使用 poll 模型
-sf	程序启动后向 PID 文件里的进程发送 FINISH 信号，这个参数需要放在命令行的最后
-st	程序启动后向 PID 文件里的进程发送 TERMINATE 信号，这个参数放在命令行的最后，经常用于重启 HAProxy 进程

如果要关闭 HAProxy，执行如下命令即可：

```
[root@haproxy-server haproxy]# killall -9 haproxy
```

如果要平滑重启 HAProxy，可执行如下命令：

```
[root@haproxy-server haproxy]# /usr/local/haproxy/sbin/haproxy -f \
> /usr/local/haproxy/conf/haproxy.cfg -st `cat /usr/local/haproxy/logs/haproxy.pid`
```

有时候为了管理和维护方便，也可以把 HAProxy 的启动与关闭写成一个独立的脚本，这里给出一个例子，脚本内容如下：

```
#!/bin/sh
# config:      /usr/local/haproxy/conf/haproxy.cfg
# pidfile:     /usr/local/haproxy/logs/haproxy.pid

# Source function library.
. /etc/rc.d/init.d/functions

# Source networking configuration.
. /etc/sysconfig/network

# Check that networking is up.
[ "$NETWORKING" = "no" ] && exit 0

config="/usr/local/haproxy/conf/haproxy.cfg"
```

```

exec="/usr/local/haproxy/sbin/haproxy"
prog=$(basename $exec)

[ -e /etc/sysconfig/$prog ] && . /etc/sysconfig/$prog

lockfile=/var/lock/subsys/haproxy

check() {
    $exec -c -V -f $config
}

start() {
    $exec -c -q -f $config
    if [ $? -ne 0 ]; then
        echo "Errors in configuration file, check with $prog check."
        return 1
    fi

    echo -n "$Starting $prog: "
    # start it up here, usually something like "daemon $exec"
    daemon $exec -D -f $config -p /usr/local/haproxy/logs/$prog.pid
    retval=$?
    echo
    [ $retval -eq 0 ] && touch $lockfile
    return $retval
}

stop() {
    echo -n "$Stopping $prog: "
    # stop it here, often "killproc $prog"
    killproc $prog
    retval=$?
    echo
    [ $retval -eq 0 ] && rm -f $lockfile
    return $retval
}

restart() {
    $exec -c -q -f $config
    if [ $? -ne 0 ]; then
        echo "Errors in configuration file, check with $prog check."
        return 1
    fi
    stop
    start
}

reload() {
    $exec -c -q -f $config

```

```

if [ $? -ne 0 ]; then
    echo "Errors in configuration file, check with $prog check."
    return 1
fi
echo -n "$"Reloading $prog: "
$exec -D -f $config -p /usr/local/haproxy/logs/$prog.pid -sf $(cat /usr/local/
haproxy/logs/$prog.pid)
retval=$?
echo
return $retval
}

force_reload() {
    restart
}

fdr_status() {
    status $prog
}

case "$1" in
    start|stop|restart|reload)
        $1
        ;;
    force-reload)
        force_reload
        ;;
    checkconfig)
        check
        ;;
    status)
        fdr_status
        ;;
    condrestart|try-restart)
        [ ! -f $lockfile ] || restart
        ;;
    *)
        echo $"Usage: $0 {start|stop|status|checkconfig|restart|try-restart|reload|force-reload}"
        exit 2
esac

```

esac

将此脚本命名为 haproxy，放在系统的 /etc/init.d/ 目录下，下面是此脚本的用法：

```

[root@haproxy-server logs]# /etc/init.d/haproxy
Usage:/etc/init.d/haproxy {start|stop|status|checkconfig|restart|try-restart|
reload|force-reload}

```

HAProxy 启动后，就可以测试 HAProxy 所实现的各种功能了。

12.3.2 测试 HAProxy 实现虚拟主机和负载均衡功能

首先通过不同 IP 的客户端以 `www.tb.com` 或者 `tb.com` 域名访问网站，如果 HAProxy 运行正常，并且 ACL 规则设置正确，`server_www` 的三台后端服务器默认的 Web 页面信息将会依次出现，这说明 HAProxy 对电商网站实现了负载均衡，同时，不会出现其他后端服务器的默认 Web 页面信息，说明 ACL 规则生效，实现虚拟主机功能。

同理，当通过不同 IP 的客户端以 `bbs.tb.com` 访问网站时，`server_bbs` 的两台后端服务器默认的 Web 页面信息将轮换出现。这表示实现了论坛的负载均衡功能，同时，不会出现其他后端服务器的默认 Web 页面信息，说明 ACL 规则生效，实现虚拟主机功能。

用同样的方法可以验证 `blog.tb.com` 是否实现了虚拟主机功能以及负载均衡功能。最后，当通过 HAProxy 服务器的 IP 或者其他方式访问时，访问请求将被调度到 `server_default` 指定的两台后端真实服务器上。

12.3.3 测试 HAProxy 的故障转移功能

测试 HAProxy 的故障转移功能也非常简单，这里假定将 `server_www` 组的一台后端服务器 `192.168.88.82` 的 `httpd` 服务停止，那么当通过 `www.tb.com` 或者 `tb.com` 域名访问网站时，这个失效的节点将不会被访问到，因为当 `httpd` 服务被停止后，HAProxy 通过 `httpchk` 方式将立刻检测到此节点无法返回数据，从而屏蔽此节点对外提供服务的功能，这样就实现了故障转移功能。通过类似的方法可以测试其他节点的应用。

12.3.4 使用 HAProxy 的 Web 监控平台

虽然 HAProxy 实现了服务的故障转移功能，但是在主机或者服务出现故障的时候，并不能发出通知告知运维人员，这对于及时性要求很高的业务系统来说，是非常不便的。不过，HAProxy 似乎也考虑到了这一点，在新的版本中 HAProxy 推出了一个基于 Web 的监控平台，通过这个平台可以查看此集群系统所有后端服务器的运行状态，在后端服务或服务器出现故障时，监控页面会通过不同的颜色来展示故障信息，这在很大程度上解决了后端服务器故障报警的问题，运维人员可通过监控这个页面来第一时间发现节点故障，进而修复故障，如图 12-4 所示。

在这个监控页面中，详细记录了 HAProxy 中配置的 `frontend`、`backend` 等信息。在 `backend` 中有各台后端真实服务器的运行状态，正常情况下，所有后端服务器都以浅绿色展示，当某台后端服务器出现故障时，将以深橙色显示。其实每种颜色代表什么状态，在上面这个图中都有详细说明。

HAProxy

Statistics Report for pid 14434

> General process information

pid = 14434 (process #1, nproc=1)
 uptime = 0s 0m0s0ms
 system limits: memmax = unlimited ulimit-n = 8213
 maxsock = 5210; maxconn = 4096; maxpipes = 0
 current conns = 1; current pipes = 0/0
 Running tasks: 1/10

active UP backup UP
 active UP going down backup UP going down
 active DOWN going up backup DOWN going up
 active or backup DOWN not checked
 active or backup DOWN for maintenance (MAINT)

Note: UP=1h last-balancing disabled is reported as "NOLEP".

Display option:

- Hide DOWN servers
- Disable refresh
- Refresh now
- CSV export

External resources:

- Primary file
- Updates (v1.4)
- Online manual



admin_stats

	Queue			Session rate			Sessions				Bytes		Denied		Errors		Warnings		Status	Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out	Req	Resp	Req	Conn	Resp		Retr	Redis	Weight	Act	Bck	Chk	Down	Downtime	Throttle
Frontend			1	12	-	1	2	2 000	66			26 130	900 800	0	0	0	0	0	0	0	OPEN							
Backend	0	0	0	7	-	0	1	2 000	28	0	26 130	900 800	0	0	0	0	23	0	0	0	3n85s UP	0	0	0		0		

statsw

	Queue			Session rate			Sessions				Bytes		Denied		Errors		Warnings		Server											
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Weight	Act	Bck	Chk	Don	Downtime	Throttle	
Frontend				0	27	-	0	5	2 000	557		227 351	157 728	0	0	0	0	0	0	0	OPEN									



server_default

		Queue			Session rate			Sessions				Bytes		Denied		Errors		Warnings		Status	Server									
		Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out	Req	Resp	Req	Conn	Resp		Retr	Redis	Weight	Act	Bck	Chk	Down	Downtime	Throttle	
	default1	0	0	-	0	12	-	0	2	-	24	24	9 995	6 892	0	0	0	0	0	0	0	3n85s UP	L7OK/200 in 16ms	6	Y	-	0	0	0s	-
	default2	0	0	-	0	12	-	0	2	-	23	23	9 546	6 677	0	0	0	0	0	0	0	3n85s UP	L7OK/200 in 8ms	6	Y	-	0	0	0s	-
	Backend	0	0	-	0	24	-	0	3	0	47	47	19 633	13 469	0	0	0	0	0	0	0	3n85s UP		12	2	0	0	0	0s	-

Choose the action to perform on the checked servers:

Apply


server_www

		Queue			Session rate			Sessions				Bytes		Denied		Errors		Warnings		Status	LastChk	Server									
		Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out	Req	Resp	Req	Conn	Resp			Retr	Redis	Weight	Act	Bck	Chk	Down	Downtime	Throttle	
	www1	0	0	-	0	9	-	0	2	-	119	119	49 221	33 517	0	0	0	0	0	0	0	3n85s	UP	L7OK/200 in 52ms	6	Y	-	0	0	0s	-
	www2	0	0	-	0	9	-	0	2	-	118	118	47 122	33 482	0	0	0	0	0	0	0	3n85s	UP	L7OK/200 in 229ms	6	Y	-	0	0	0s	-
	www3	0	0	-	0	9	-	0	2	-	118	118	48 594	33 278	0	0	0	0	0	0	0	3n85s	UP	L7OK/200 in 15ms	6	Y	-	0	0	0s	-
	Backend	0	0	-	0	27	-	0	3	0	355	355	144 937	100 257	0	0	0	0	0	0	0	3n85s	UP		18	3	0	0	0	0s	-

Choose the action to perform on the checked servers:

Apply



server_bbs

	Queue			Session rate			Sessions					Bytes		Denied		Errors		Warnings		Status	Server							
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out	Req	Resp	Req	Conn	Resp	Retr		Redis	Weight	Act	Bck	Chk	Down	Downtime	Throttle
	bbs1	0	0	-	0	11	-	0	2	-	34	34	14 278	9 795	0	0	0	0	0	0	3n85s	L7OK/200 in 15ms	6	Y	-	0	0	0s
	bbs2	0	0	-	0	12	-	0	1	-	33	33	13 055	9 490	0	0	0	0	0	0	3n85s	L7OK/200 in 279ms	6	Y	-	0	0	0s
	Backend	0	0	-	0	23	-	0	2	0	67	67	27 833	19 211	0	0	0	0	0	0	3n85s		12	2	0	0	0	0s

Choose the action to perform on the checked servers:

Apply

server_blog

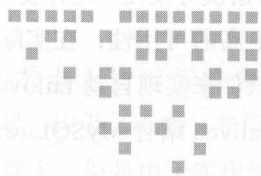
	Queue			Session rate			Sessions				Bytes		Denied		Errors		Warnings		Status	LastChk	Server							
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out	Req	Resp	Req	Conn	Resp			Retr	Redis	Weight	Act	Bck	Chk	Down	Downtime
	bbg1	0	0	-	0	13	-	0	2	-	43	43	17 516	12 194	0	0	0	0	0	0	3n85s UP	L7OK/200 in 23ms	6	Y	-	0	0	0s
	bbg2	0	0	-	0	14	-	0	1	-	42	42	17 528	12 036	0	0	0	0	0	0	3n85s UP	L7OK/200 in 65ms	6	Y	-	0	0	0s
	Backend	0	0	-	0	27	-	0	3	0	85	85	35 044	24 230	0	0	0	0	0	0	3n85s UP		12	2	0		0	0s

Choose the action to perform on the checked servers:

Apply

图 12-4 HAProxy 的 Web 监控页面

在这个监控页面中，还可以执行关闭自动刷新、隐藏故障状态的节点、手动刷新、导出数据为 CSV 文件等各种操作。在新版的 HAProxy 中，又增加了对 backend 后端节点的管理功能，例如，可以在 Web 页面下执行 Disable、Enable、Soft Stop、Soft Start 等对后端节点的管理操作。这个功能在后端节点升级、故障维护时非常有用。



构建高性能的 MySQL 集群系统

13.1 常见的高可用 MySQL 解决方案

数据库作为最基础的数据存储服务之一，在存储系统中有着非常重要的地位，因此要求其具备高可用性无可厚非。能实现不同 SLA（服务水平协定）的解决方案有很多种，这些方案可以保证数据库服务器在硬件或软件出现故障时服务继续可用。

高可用性需要解决的主要问题有两个，一个是如何实现数据共享或同步数据，另一个是如何处理 failover。数据共享一般的解决方案是通过 SAN (Storage Area Network) 来实现，而数据同步可以通过 rsync 软件或 DRBD 技术来实现。failover 的意思是指当服务器死机或出现错误时可以自动切换到其他备用服务器，不影响服务器上业务系统的运行。本章重点介绍目前比较成熟的 MySQL 高可用解决方案。

13.1.1 主从复制解决方案

这是 MySQL 自身提供的一种高可用解决方案，数据同步方法采用的是 MySQL replication 技术。MySQL replication 技术就是一个日志的复制过程，在复制过程中一台服务器充当主服务器，而一台或多台其他服务器充当从服务器，简单说就是，从服务器到主服务器拉取二进制日志文件，然后再将日志文件解析成相应的 SQL 在从服务器上重新执行一遍主服务器的操作，通过这种方式保证数据的一致性。

MySQL replication 技术仅仅提供了日志的同步执行功能，而从服务器只能提供读操作，并且当主服务器发生故障时，必须手动处理 failover，通常的做法是将一台从服务器更改为

主服务器。这种解决方案在一定程度上实现了 MySQL 的高可用性，可以实现 90.000% 的 SLA。

为了达到更高的可用性，在实际的应用环境中，一般都采用 MySQL replication 技术配合高可用集群软件来实现自动 failover，这种方式可以实现 95.000% 的 SLA。13.2 节会重点介绍通过 Keepalived 结合 MySQL replication 技术实现 MySQL 高可用架构的解决方案。

13.1.2 MMM 高可用解决方案

MMM (Master-Master Replication Manager for MySQL) MySQL 主主复制管理器，提供了 MySQL 主主复制配置的监控、故障转移和管理的一套可伸缩的脚本套件。在 MMM 高可用解决方案中，典型的应用是双主多从架构，通过 MySQL replication 技术可以实现两台服务器互为主从，且在任何时候只有一个节点可以写入，避免多点写入的数据冲突。同时，当可写的主节点故障时，MMM 套件可以立刻监控到，然后将服务自动切换到另一个主节点，继续提供服务，从而实现 MySQL 的高可用。

MMM 方案是目前比较成熟的 MySQL 高可用解决方案，可以实现 99.000% 的 SLA。13.3 节会重点介绍通过 MMM 实现 MySQL 高可用解决方案。

13.1.3 Heartbeat/SAN 高可用解决方案

此解决方案是借助第三方软硬件实现的，在这个方案中，处理 failover 的方式是高可用集群软件 Heartbeat，它监控和管理各个节点间连接的网络，并监控集群服务，当节点出现故障或者服务不可用时，自动在其他节点启动集群服务。

在数据共享方面，通过 SAN 存储来共享数据，在正常状态下，集群主节点将挂载存储进行数据读写，而当集群发生故障时，Heartbeat 会首先通过一个仲裁设备将主节点挂载的存储设备释放，然后在备用节点上挂载存储，接着启动服务，通过这种方式实现数据的共享和同步。这种数据共享方式实现简单，但是成本较高，并且存在脑裂的可能，需要根据实际应用环境来选择。这种方案可以实现 99.990% 的 SLA。

13.1.4 Heartbeat/DRBD 高可用解决方案

这种高可用解决方案也是借助第三方软硬件实现的，在处理 failover 的方式上依旧采用 Heartbeat，不同的是，在数据共享方面，采用了基于块级别的数据同步软件 DRBD 来实现。

DRBD (Distributed Replicated Block Device) 是一个用软件实现的、无共享的、服务器之间镜像块设备内容的存储复制解决方案。和 SAN 网络不同，它并不共享存储，而是通过服务器之间的网络复制数据。这种方案实现起来稍微复杂，同时也存在脑裂的问题，可以实现 99.900% 的 SLA。

13.1.5 MySQL Cluster 高可用解决方案

MySQL Cluster 由一组服务节点构成，每个服务节点上均运行着多种进程，包括 MySQL 服务器、NDB Cluster 的数据节点、管理服务器，以及（可能）专门的数据访问程序。此解决方案是 MySQL 官方主推的技术方案，功能强大，但是由于实现较为繁琐，配置麻烦，企业实际应用并不多。MySQL Cluster 的标准版和电信版可以达到 99.999% 的 SLA。

13.2 通过 Keepalived 搭建 MySQL 双主模式的高可用集群系统

13.2.1 MySQL Replication 介绍

MySQL Replication 是 MySQL 自身提供的一个主从复制功能，其实也就是一台 MySQL 服务器（称为 Slave）从另一台 MySQL 服务器（称为 Master）上复制日志，然后解析日志并应用到自身的过程。MySQL Replication 是单向、异步复制，基本复制过程为：Master 服务器首先将更新写入二进制日志文件，并维护文件的一个索引以跟踪日志的循环。这些日志文件可以发送到 Slave 服务器进行更新。当一台 Slave 服务器连接 Master 服务器时，它从 Master 服务器日志中读取上一次成功更新的位置。然后 Slave 服务器开始接收从上一次完成更新后发生的所有更新，所有更新完成，将等待主服务器通知新的更新。

MySQL Replication 支持链式复制，也就是说 Slave 服务器下还可以再链接 Slave 服务器，同时 Slave 服务器也可以充当 Master 服务器角色。这里需要注意的是，在 MySQL 主从复制中，所有表的更新必须在 Master 服务器上进行，Slave 服务器仅提供查询操作。

基于单向复制的 MySQL Replication 技术有如下优点：

- 增加了 MySQL 应用的健壮性，如果 Master 服务器出现问题，可以随时切换到 Slave 服务器，继续提供服务。
- 可以将 MySQL 读、写操作分离，写操作只在 Master 服务器完成，读操作可在多个 Slave 服务器上完成，由于 Master 服务器和 Slave 服务器是保持数据同步的，因此不会对前端业务系统产生影响。同时，通过读、写的分离，可以大大降低 MySQL 的运行负荷。
- 在网络环境较好，业务量不是很大的环境中，Slave 服务器同步数据非常快，基本可以达到实时同步，并且，Slave 服务器在同步过程中不会干扰 Master 服务器。

MySQL Replication 支持多种类型的复制方式，常见的有基于语句的复制、基于行的复制和混合类型的复制。下面分别进行介绍。

(1) 基于语句的复制

MySQL 默认采用基于语句的复制，效率很高。基本方式是：在 Master 服务器上执行的 SQL 语句，在 Slave 服务器上再次执行同样的语句。而一旦发现没法精确复制时，会自动选择基于行的复制。

(2) 基于行的复制

基本方式为：把 Master 服务器上改变的内容复制过去，而不是把 SQL 语句在从服务器上执行一遍，从 MySQL 5.0 开始支持基于行的复制。

(3) 混合类型的复制

其实就是上面两种类型的组合，默认采用基于语句的复制，如果发现基于语句的复制无法精确完成，就会采用基于行的复制。

13.2.2 MySQL Replication 实现原理

MySQL Replication 是一个从 Master 复制到一台或多台 Slave 的异步过程，在 Master 与 Slave 之间实现整个复制过程主要由三个线程来完成，其中一个 IO 线程在 Master 端，另两个线程（SQL 线程和 IO 线程）在 Slave 端。

要实现 MySQL Replication，首先在 Master 服务器上打开 MySQL 的 Binary Log（产生二进制日志文件）功能，因为整个复制过程实际上就是 Slave 从 Master 端获取该日志，然后在自身上将二进制文件解析为 SQL 语句并完全顺序地执行 SQL 语句所记录的各种操作。更详细的过程如下。

1) 首先 Slave 上的 IO 线程连接上 Master，然后请求从指定日志文件的指定位置或者从最开始的日志位置之后的日志内容。

2) Master 在接收到来自 Slave 的 IO 线程请求后，通过自身的 IO 线程，根据请求信息读取指定日志位置之后的日志信息，并返回给 Slave 端的 IO 线程。返回信息中除了日志所包含的信息之外，还包括此次返回的信息在 Master 端对应的 Binary Log 文件的名称以及在 Binary Log 中的位置。

3) Slave 的 IO 线程接收到信息后，将获取到的日志内容依次写入 Slave 端的 Relay Log 文件（类似于 mysql-relay-bin.xxxxxx）的最后，并且将读取到的 Master 端的 Binary Log 的文件名和位置记录到一个名为 master-info 的文件中，以便在下一次读取的时候能够迅速定位开始往后读取日志信息的位置。

4) Slave 的 SQL 线程在检测到 Relay Log 文件中新增加了内容后，会马上解析该 Relay Log 文件中的内容，将日志内容解析为 SQL 语句，然后在自身执行这些 SQL，由于是在 Master 端和 Slave 端执行了同样的 SQL 操作，所以两端的数据是完全一样的。至此整个复制过程结束。

13.2.3 MySQL Replication 常用架构

MySQL Replication 技术在实际应用中有多种实现架构，常见的有：

- 一主一从，即一台 Master 服务器和一台 Slave 服务器。这是最常见的架构。
- 一主多从，即一台 Master 服务器和两台或两台以上 Slave 服务器。经常用在写操作不频繁、查询量比较大的业务环境中。
- 主主互备，又称双主互备，即两台 MySQL Server 互相将对方作为自己的 Master，自己又同时作为对方的 Slave 来进行复制。主要用于对 MySQL 写操作要求比较高的环境中，避免了 MySQL 单点故障。在 13.2.5 节会重点介绍此种架构的实现方式。
- 双主多从，其实就是双主互备，然后再加上多台 Slave 服务器。主要用于对 MySQL 写操作要求比较高，同时查询量比较大的环境中。这种构架在 13.3 节会做更详细的介绍。

其实可以根据具体的情况灵活地将 Master/Slave 结构进行变化组合，但万变不离其宗，在进行 MySQL Replication 的各种部署之前，必须遵守的规则如下：

- 同一时刻只能有一台 Master 服务器进行写操作。
- 一台 Master 服务器可以有多台 Slave 服务器。
- 无论是 Master 服务器还是 Slave 服务器，都要确保各自的 Server ID 唯一，否则双主互备就会出问题。
- 一台 Slave 服务器可以将其从 Master 服务器获得的更新信息传递给其他的 Slave 服务器。依此类推。

13.2.4 MySQL 主主互备模式架构

企业级 MySQL 集群具备高可用、可扩展、易管理、低成本的特点。下面将介绍企业环境中经常应用的一个解决方案，即 MySQL 的双主互备架构，主要设计思路是通过 MySQL Replication 技术将两台 MySQL Server 互相将对方作为自己的 Master，自己又同时作为对方的 Slave 来进行复制。这样就实现了高可用架构中的数据同步功能，同时，将采用 Keepalived 来实现 MySQL 的自动 failover。在这种架构中，虽然两台 MySQL Server 互为主从，但同一时刻只有一台 MySQL Server 可读写，而另一台 MySQL Server 只能进行读操作，这样可保证数据的一致性。整个架构如图 13-1 所示。

在图 13-1 中，DB1 和 DB2 互为主从，这样就保证了两台 MySQL 的数据始终是同步的，同时在 DB1 和 DB2 上还需要安装高可用软件 Keepalived。在正常情况下，Web Server 主机仅从 DB1 进行数据的读、写操作，DB2 只负责从 DB1 同步数据。而 Keepalived 维护着一个 VIP，此 IP 用来对外提供连接服务，同时，Keepalived 还负责监控 DB1 和 DB2 上 MySQL 数据库的运行状态，当 DB1 主机出现故障或 MySQL 运行异常时，自动将 VIP 地址和 MySQL 服务切换到

DB2 上，此时 Web Server 主机继续从 DB2 进行数据的读、写操作。通过 Keepalived 保持了数据库服务的连续性，整个切换过程非常快，并且对前端 Web Server 主机是透明的。

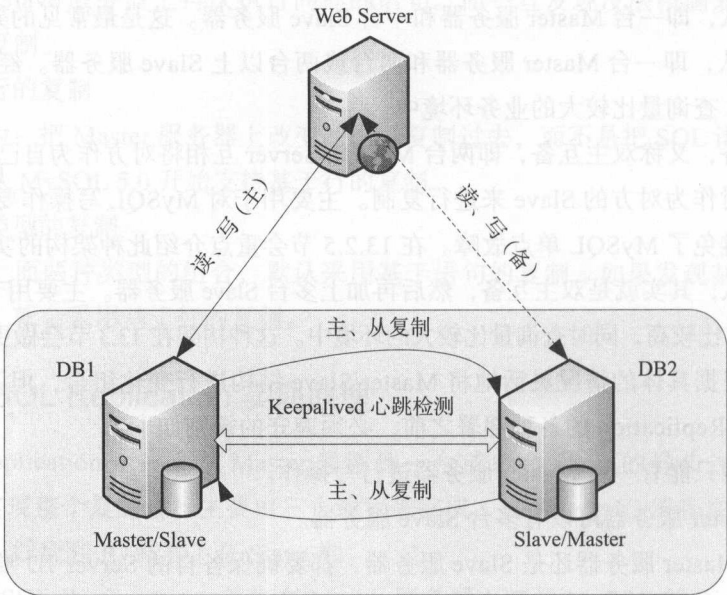


图 13-1 MySQL 双主互备架构

13.2.5 MySQL 主主互备模式配置

MySQL 主从复制的配置还是比较简单的，仅仅需要修改 MySQL 配置文件即可，这里要配置的是主主互备模式，但配置过程和一主一从结构是完全一样的，配置环境如表 13-1 所示。

表 13-1 MySQL 主主互备模式配置环境

主机名	操作系统版本	MySQL 版本	主机 IP	MySQL VIP
DB1 (MASTER)	CentOS release 6.3	mysql-5.1.73	192.168.88.11	192.168.88.10
DB2 (Slave)	CentOS release 6.3	mysql-5.1.73	192.168.88.12	

下面开始进入配置过程。

1. 修改 MySQL 配置文件

在默认情况下，MySQL 的配置文件是 /etc/my.cnf，首先修改 DB1 主机的配置文件，在 /etc/my.cnf 文件中的 “[mysqld]” 段添加如下内容：

```
server-id = 1
log-bin=mysql-bin
relay-log = mysql-relay-bin
```

```

replicate-wild-ignore-table=mysql.%
replicate-wild-ignore-table=test.%
replicate-wild-ignore-table=information_schema.%

```

然后修改 DB2 主机的配置文件，在 /etc/my.cnf 文件中的 “[mysqld]” 段添加如下内容：

```

server-id = 2
log-bin=mysql-bin
relay-log = mysql-relay-bin
replicate-wild-ignore-table=mysql.%
replicate-wild-ignore-table=test.%
replicate-wild-ignore-table=information_schema.%

```

其中，server-id 是节点标识，主、从节点不能相同，必须全局唯一。log-bin 表示开启 MySQL 的 binlog 日志功能。“mysql-bin” 表示日志文件的命名格式，会生成文件名为 mysql-bin.000001、mysql-bin.000002 等的日志文件。relay-log 用来定义 relay-log 日志文件的命名格式。replicate-wild-ignore-table 是个复制过滤选项，可以过滤不需要复制的数据库或表，例如 “mysql.%” 表示不复制 MySQL 库下的所有对象，其他依此类推。与此对应的是 replicate_wild_do_table 选项，用来指定需要复制的数据库或表。

这里需要注意的是，不要在主库上使用 binlog-do-db 或 binlog-ignore-db 选项，也不要再从库上使用 replicate-do-db 或 replicate-ignore-db 选项，因为这样可能产生跨库更新失败的问题。推荐在从库上使用 replicate_wild_do_table 和 replicate-wild-ignore-table 两个选项来解决复制过滤问题。

2. 手动同步数据库

如果 DB1 上已经有 MySQL 数据，那么在执行主主互备之前，需要将 DB1 和 DB2 上两个 MySQL 的数据保持同步，首先在 DB1 上备份 MySQL 数据，执行如下 SQL 语句：

```

mysql>FLUSH TABLES WITH READ LOCK;
Query OK, 0 rows affected (0.00 sec)

```

不要退出这个终端，否则这个锁就失效了。在不退出终端的情况下，再开启一个终端直接打包压缩数据文件或使用 mysqldump 工具导出数据。这里通过打包 mysql 文件来完成数据的备份，操作过程如下：

```

[root@DB1 ~]# cd /var/lib/
[root@DB1 lib]# tar zcvf mysql.tar.gz mysql
[root@DB1 lib]# scp mysql.tar.gz DB2:/var/lib/

```

将数据传输到 DB2 后，依次重启 DB1 和 DB2 上面的 MySQL。

3. 创建复制用户并授权

首先在 DB1 的 MySQL 库中创建复制用户，操作过程如图 13-2 所示。


```
mysql> grant replication slave on *.* to 'repl_user'@'192.168.88.12' identified by 'repl_passwd';
mysql> show master status;
```

File	Position	Binlog_Do_DB	Binlog_Ignore_DB
mysql-bin.000001	106		

图 13-2 在 DB1 MySQL 库中创建复制用户

然后在 DB2 的 MySQL 库中将 DB1 设为主服务器，操作过程如图 13-3 所示。

这里需要注意 `master_log_file` 和 `master_log_pos` 两个选项，这两个选项的值刚好是在 DB1 上通过 SQL 语句“`show master status`”查询到的结果。

接着就可以在 DB2 上启动 slave 服务了，可执行如下 SQL 命令：

```
mysql> start slave;
```

下面查看 DB2 上 slave 的运行状态，如图 13-4 所示。

```
mysql> change master to \
-> master host='192.168.88.11',
-> master user='repl_user',
-> master password='repl_passwd',
-> master_log_file='mysql-bin.000001',
-> master_log_pos=106;
```

图 13-3 在 DB2 MySQL 库中将 DB1 设为主服务器

```
mysql> show slave status\G;
***** 1. row *****
Slave_IO_State: Waiting for master to send event
Master Host: 192.168.88.11
Master User: repl_user
Master Port: 3306
Connect Retry: 60
Master_Log_File: mysql-bin.000001
Read_Master_Log_Pos: 106
Relay_Log_File: mysql-relay-bin.000001
Relay_Log_Pos: 251
Relay_Master_Log_File: mysql-bin.000001
Slave_IO_Running: Yes
Slave_SQL_Running: Yes
Replicate_Do_DB:
Replicate_Ignore_DB:
Replicate_Do_Table:
Replicate_Ignore_Table:
Replicate_Wild_Do_Table:
Replicate_Wild_Ignore_Table: mysql.%,test.%,information_schema.%
Last_Errno: 0
Last_Error:
Skip_Counter: 0
Exec_Master_Log_Pos: 106
Relay_Log_Space: 919
Until_Condition: None
Until_Log_File:
Until_Log_Pos: 0
Master_SSL_Allowed: No
Master_SSL_CA_File:
Master_SSL_CA_Path:
Master_SSL_Cert:
Master_SSL_Cipher:
Master_SSL_Key:
Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
Last_IO_Errno: 0
Last_IO_Error:
Last_SQL_Errno: 0
Last_SQL_Error:
1 row in set (0.00 sec)
```

图 13-4 DB2 上 slave 的运行状态

通过查看 slave 的运行状态可以发现，一切运行正常，这里需要重点关注的是 Slave_IO_Running 和 Slave_SQL_Running，这两个就是在 Slave 节点上运行的主从复制线程，正常情况下这两个值都应该为 Yes。另外，还需要注意的是 Slave_IO_State、Master_Host、Master_Log_File、Read_Master_Log_Pos、Relay_Log_File、Relay_Log_Pos 和 Relay_Master_Log_File 几个选项，从图中可以查看出 MySQL 复制的运行原理及执行规律。最后还有一个 Replicate_Wild_Ignore_Table 选项，这个是之前在 my.cnf 中添加过的，通过此选项的输出值可以知道过滤了哪些数据库。

到这里，从 DB1 到 DB2 的 MySQL 主从复制已经完成。接下来开始配置从 DB2 到 DB1 的 MySQL 主从复制，这个配置过程与上面的过程完全一样，首先在 DB2 的 MySQL 库中创建复制用户，操作如图 13-5 所示。

```
mysql> grant replication slave on *.* to 'repl_user'@'192.168.88.11' identified by 'repl_passwd';
mysql> show master status;
```

File	Position	Binlog_Do_DB	Binlog_Ignore_DB
mysql-bin.000001	106		

图 13-5 在 DB2 MySQL 库中创建复制用户

然后在 DB1 的 MySQL 库中将 DB2 设为主服务器，操作如图 13-6 所示。

```
mysql> change master to \
-> master_host='192.168.88.12',
-> master_user='repl_user',
-> master_password='repl_passwd',
-> master_log_file='mysql-bin.000001',
-> master_log_pos=106;
```

图 13-6 在 DB1 MySQL 库中将 DB2 设为主服务器

最后，就可以在 DB1 上启动 slave 服务了，可执行如下 SQL 命令：

```
mysql> start slave;
```

下面查看下 DB1 上 slave 的运行状态，如图 13-7 所示。

从图 13-7 中可以看出 Slave_IO_Running 和 Slave_SQL_Running 都是 Yes 状态，表明 DB1 上复制服务运行正常。至此，MySQL 双主模式的主从复制配置完毕。

13.2.6 配置 Keepalived 实现 MySQL 双主高可用

在进行高可用配置之前，首先需要在 DB1 和 DB2 服务器上安装 Keepalived 软件。Keepalived 在前面章节中已经做过详细介绍，关于 Keepalived 的安装这里不再说明，直接进

入 Keepalived 的配置过程。下面是 DB1 服务器上 /etc/keepalived/keepalived.conf 文件的内容。

```
mysql> show slave status\G;
***** 1. row *****
Slave_IO_State: Waiting for master to send event
Master_Host: 192.168.88.12
Master_User: repl_user
Master_Port: 3306
Connect_Retry: 60
Master_Log_File: mysql-bin.000001
Read_Master_Log_Pos: 106
Relay_Log_File: mysql-relay-bin.000001
Relay_Log_Pos: 251
Relay_Master_Log_File: mysql-bin.000001
Slave_IO_Running: Yes
Slave_SQL_Running: Yes
Replicate_Do_DB:
Replicate_Ignore_DB:
Replicate_Do_Table:
Replicate_Ignore_Table:
Replicate_Wild_Do_Table:
Replicate_Wild_Ignore_Table: mysql.%,test.%,information_schema.%
Last_Errno: 0
Last_Error:
Skip_Counter: 0
Exec_Master_Log_Pos: 106
Relay_Log_Space: 908
Until_Condition: None
Until_Log_File:
Until_Log_Pos: 0
Master_SSL_Allowed: No
Master_SSL_CA_File:
Master_SSL_CA_Path:
Master_SSL_Cert:
Master_SSL_Cipher:
Master_SSL_Key:
Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
Last_IO_Errno: 0
Last_IO_Error:
Last_SQL_Errno: 0
Last_SQL_Error:
1 row in set (0.00 sec)
```

图 13-7 DB1 上 slave 的运行状态

```
global_defs {
    notification_email {
        acassen@firewall.loc
        failover@firewall.loc
        sysadmin@firewall.loc
    }
    notification_email_from Alexandre.Cassen@firewall.loc
    smtp_server 192.168.200.1
    smtp_connect_timeout 30
    router_id MySQLHA_DEVEL
}

vrrp_script check_mysqld {
    script "/etc/keepalived/mysqlcheck/check_slave.pl 127.0.0.1" # 检测 MySQL 复制状态的脚本
    interval 2
    weight 21
}
```

```

vrrp_instance HA_1 {
    state BACKUP          # 在 DB1 和 DB2 上均配置为 BACKUP
    interface eth0
    virtual_router_id 80
    priority 100
    advert_int 2
    nopreempt             # 不抢占模式，只在优先级高的机器上设置即可，优先级低的机器可不设置

    authentication {
        auth_type PASS
        auth_pass qweasdzxc
    }

    track_script {
        check_mysql
    }

    virtual_ipaddress {
        192.168.88.10/24 dev eth0 #MySQL 的对外服务 IP，即 VIP
    }
}

```

其中，/etc/keepalived/mysqlcheck/check_slave.pl 文件的内容为：

```

#!/usr/bin/perl -w
use DBI;
use DBD::mysql;

# CONFIG VARIABLES
$SBM = 120;
$db = "ixdba";
$host = $ARGV[0];
$port = 3306;
$user = "root";
$pw = "xxxxxx";

# SQL query
$query = "show slave status";

$dbh = DBI->connect("DBI:mysql:$db:$host:$port", $user, $pw, { RaiseError =>
0, PrintError => 0 });

if (!defined($dbh)) {
    exit 1;
}

$sqlQuery = $dbh->prepare($query);
$sqlQuery->execute;

```



```

$Slave_IO_Running = "";
$Slave_SQL_Running = "";
$Seconds_Behind_Master = "";

while (my $ref = $sqlQuery->fetchrow_hashref()) {
    $Slave_IO_Running = $ref->{'Slave_IO_Running'};
    $Slave_SQL_Running = $ref->{'Slave_SQL_Running'};
    $Seconds_Behind_Master = $ref->{'Seconds_Behind_Master'};
}

$sqlQuery->finish;
$dbh->disconnect();

if ( $Slave_IO_Running eq "No" || $Slave_SQL_Running eq "No" ) {
    exit 1;
} else {
    if ( $Seconds_Behind_Master > $SBM ) {
        exit 1;
    } else {
        exit 0;
    }
}

```

这是用 Perl 写的检测 MySQL 复制状态的脚本，只需修改文件中 MySQL 数据库的端口、用户名和密码即可直接使用，但在使用前要保证此脚本有可执行权限。

接着将 keepalived.conf 文件和 check_slave.pl 文件复制到 DB2 服务器上对应的位置，然后将 keepalived.conf 文件中 priority 值修改为 90。由于配置的是不抢占模式，因此，还需要去掉 nopreempt 选项。

在完成所有配置后，分别在 DB1 和 DB2 上启动 keepalived 服务，在正常情况下 VIP 地址应该运行在 DB1 服务器上。

13.2.7 测试 MySQL 主从同步功能

为了验证 MySQL 的复制功能，可以编写一个简单的程序进行测试，也可以通过远程客户端登录进行测试。这里通过一个远程 MySQL 客户端，然后利用 MySQL 的 VIP 地址登录，看能否登录，并在登录后进行读、写操作，看看 DB1 和 DB2 之间能否实现数据同步。由于采用远程登录测试，因此 DB1 和 DB2 两台 MySQL 服务器都要事先做好授权，允许从远程登录。

1. 在远程客户端通过 VIP 登录测试

首先通过远程 MySQL 客户端命令行登录 VIP 为“192.168.88.10”的数据库，操作过程

如图 13-8 所示。

```
[root@apps ~]# mysql -uroot -p -h 192.168.88.10
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2513
Server version: 5.1.73-log Source distribution
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql> show variables like "%hostname%";
+-----+
| Variable_name | Value |
+-----+
| hostname      | DB1   |
+-----+
1 row in set (0.00 sec)
mysql> show variables like "%server_id%";
+-----+
| Variable_name | Value |
+-----+
| server_id     | 1     |
+-----+
1 row in set (0.00 sec)
```

图 13-8 MySQL 远程客户端 VIP 登录测试

从 SQL 输出结果看，可以通过 VIP 登录，并且登录了 DB1 服务器。

2. 数据复制功能测试

接着上面的 SQL 操作过程，通过远程的 MySQL 客户端连接 VIP，进行读、写操作测试，操作过程如图 13-9 所示。

```
mysql> create database repldb;
Query OK, 1 row affected (0.01 sec)
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql      |
| repldb     |
| test       |
+-----+
4 rows in set (0.00 sec)
mysql> use repldb;
mysql> create table repl_table(id int,email varchar(80),password varchar(40) not null);
Query OK, 0 rows affected (0.02 sec)
mysql> show tables;
+-----+
| Tables in repldb |
+-----+
| repl_table        |
+-----+
1 row in set (0.02 sec)
mysql> insert into repl table (id,email,password) values(1,"master@189.cn","qweasd");
Query OK, 1 row affected (0.00 sec)
```

图 13-9 向 DB1 数据库中插入数据

这个过程创建了一个数据库 repldb，然后在 repldb 库中创建了一张表 repl_table。为了验证

数据是否复制到 DB2 主机上，登录 DB2 主机的 MySQL 命令行，查询过程如图 13-10 所示。

```
[root@DB2 mysql]# mysql -uroot -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 6536
Server version: 5.1.73-log Source distribution

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| repldb |
| test |
+-----+
4 rows in set (0.00 sec)

mysql> use repldb;
Database changed
mysql> show tables;
+-----+
| Tables_in_repldb |
+-----+
| repl_table |
+-----+
1 row in set (0.00 sec)

mysql> select * from repl_table;
+-----+-----+-----+
| id | email | password |
+-----+-----+-----+
| 1 | master@189.cn | qweasd |
+-----+-----+-----+
1 row in set (0.02 sec)
```

图 13-10 在 DB2 数据库中查询数据是否已经同步

从 SQL 输出结果看，刚才创建的库和表都已经同步到了 DB2 服务器上。其实也可以直接登录 DB2 服务器，然后执行数据库的读、写操作，看数据能否迅速同步到 DB1 的 MySQL 数据库中。测试过程与前面的完全一样，这里不再重复介绍。

13.2.8 测试 Keepalived 实现 MySQL 故障转移

为了测试 Keepalived 实现的故障转移功能，需要模拟一些故障，比如，可以通过断开 DB1 主机的网络、关闭 DB1 主机、关闭 DB1 上 MySQL 服务等各种操作实现。这里在 DB1 服务器上关闭 MySQL 的日志接收功能，以此来模拟 DB1 上 MySQL 的故障。由于在 DB1 和 DB2 服务器上都添加了监控 MySQL 运行状态的脚本 check_slave.pl，因此当关闭 DB1 的 MySQL 日志接收功能后，Keepalived 会立刻检测到，接着执行切换操作。测试过程如下。

1. 停止 DB1 服务器的日志接收功能

首先在远程 MySQL 客户端以 VIP 地址登录 MySQL 系统，不要退出这个连接，然后在

DB1 服务器的 MySQL 命令行执行如下操作：

```
mysql> slave stop;
```

2. 在远程客户端测试

继续在刚才打开的远程 MySQL 连接中执行命令，操作过程如图 13-11 所示。

```
mysql> select * from repldb.repl_table;
ERROR 2013 (HY000): Lost connection to MySQL server during query
mysql> select * from repldb.repl_table;
ERROR 2006 (HY000): MySQL server has gone away
No connection. Trying to reconnect...
Connection id: 39063
Current database: repldb
+-----+-----+-----+
| id | email | password |
+-----+-----+-----+
| 1 | master@189.cn | qweasd |
+-----+-----+-----+
1 row in set (0.14 sec)
mysql> show variables like "%hostname%";
+-----+-----+-----+
| Variable_name | Value |
+-----+-----+-----+
| hostname | DB2 |
+-----+-----+-----+
1 row in set (0.00 sec)
mysql> show variables like "%server_id%";
+-----+-----+-----+
| Variable_name | Value |
+-----+-----+-----+
| server_id | 2 |
+-----+-----+-----+
1 row in set (0.02 sec)
```

图 13-11 通过 Keepalived 实现 MySQL 故障转移测试

从这个操作过程可以看出，在 Keepalived 切换后，之前的 session 连接失效，所以第一个查询命令失败。然后重新执行查询命令，MySQL 会执行重新连接，随后输出了查询结果，从后面两个 SQL 的查询结果可知，MySQL 服务已经从 DB1 服务器切换到 DB2 服务器。Keepalived 的切换过程非常迅速，整个过程大概持续 1~3s，重新切换到新的服务器后，之前所有的 MySQL 连接失效，重新连接可以恢复正常。

接着，重新打开 DB1 上 MySQL 的日志接收功能，可以发现 Keepalived 将不再执行切换操作，因为上面将 Keepalived 配置为不抢占模式，此时，MySQL 服务将一直在 DB2 服务器上运行，直到 DB2 主机或服务出现故障才再次进行切换操作。这样做的原因是在数据库环境下，每次切换的代价很大，因而关闭了 Keepalived 的主动抢占模式。

13.3 通过 MMM 构建 MySQL 高可用集群系统

13.2 节介绍了双主互备的 MySQL 高可用解决方案, 此方案通过 MySQL 的主从复制功能实现了 MySQL 集群的数据同步功能, 同时通过 Keepalived 实现任意节点故障后自动切换到另一个节点的故障转移功能。这个架构非常适用于对 MySQL 持续服务要求比较高的场合, 但是此方案也有一些缺陷: 首先, 任意时刻只有一个节点对外提供服务, 另一个节点处于数据同步状态, 不对外提供任何服务, 这在一定程度上浪费了资源; 其次, MySQL 主从复制支持一主多从架构, 这对于读操作比较大的 Web 业务系统来说, 可以将读的压力分散到多个 Slave 上, 而双主互备架构显然没有充分发挥 MySQL 复制的优势; 最后, 如果在双主互备的基础上增加多个 Slave 节点, 将会出现问题, 因为在 Master 节点切换到备用 Master 节点后, 多个 Slave 节点的 “Master_Host” 无法自动切换到备用 Master 节点, 从而导致整个 MySQL 高可用架构出现问题。

要解决这些问题其实并不难, 通过 MMM 集群套件 (MySQL 主主复制管理器) 即可轻松实现。本节将重点介绍通过 MMM 集群套件来构建高性能的 MySQL 集群系统。

13.3.1 MMM 高可用 MySQL 方案简介

MMM 高可用 MySQL 方案已经在本章开始做过一个简单的介绍, 它是一个通过 Perl 编写的、基于 MySQL 主从复制的、成熟完善的 MySQL 高可用集群解决方案, 由一个管理端 (monitor) 和多个代理端 (agent) 构成。通过 MMM 可以实现监控和管理 MySQL 主主复制和服务状态, 同时也可以监控多个 Slave 节点的复制以及运行状态, 并且可以做到任意节点发生故障时实现自动切换的功能。MMM 集群方案的出现为 MySQL 的读、写分离架构的应用提供了一个很好的平台, 这是因为 MMM 集群方案将读 IP (reader IP) 和写 IP (writer IP) 从数据库层面提取出来, 在业务系统只需调用即可实现读、写分离功能。

虽然 MMM 是一个 MySQL 主主复制管理器, 但是在整个集群中, 同一时刻只有一个 Master 是可写的, 这样做是为了保证数据的完整性和安全性。

MMM 套件主要的功能是通过下面三个脚本来实现的。

❑ mmm_mond

这是一个监控进程, 运行在管理节点上, 主要负责对所有数据库的监控工作, 同时决定和处理所有节点的角色切换。

❑ mmm_agentd

这是一个代理进程, 运行在每台 MySQL 服务器上, 主要完成监控的测试工作以及执行简单的远端服务设置。

❑ mmm_control

这是一个简单的管理脚本，用来查看和管理集群运行状态，同时管理 mmm_mond 进程。

MMM 集群套件具有良好的稳定性、高可用性和可扩展性，当活动的 Master 服务器故障以后，备用 Master 服务器会立刻接管，而其他的 Slave 服务器也能自动切换到备用 Master 服务器继续进行同步复制，整个过程无需人为干预。

当然，MMM 集群套件也有一定的缺点：首先 MMM 架构需要多个节点、多个 IP，对服务器数量有要求；其次，MMM 方案在读、写非常繁忙的业务系统下表现不是很稳定，可能会出现复制延时、切换失效等问题。因此，MMM 方案并不太适应于对数据安全性要求很高，并且读、写频繁的环境中。

13.3.2 MMM 典型应用方案

MMM 有多种应用架构，最简单的是两个节点的运行环境，如图 13-12 所示。

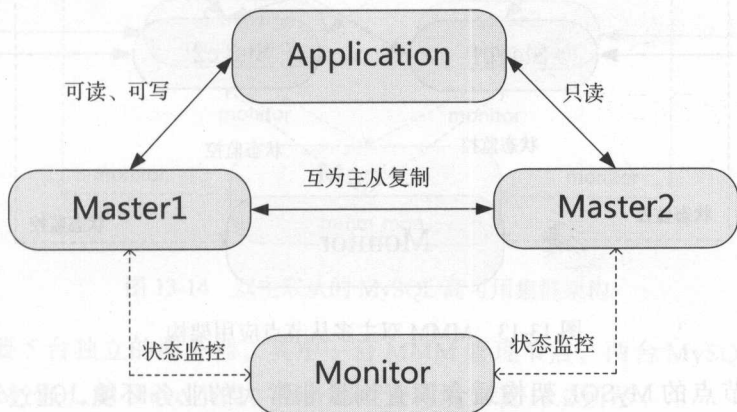


图 13-12 MMM 双 Master 节点应用架构

双 Master 节点的 MySQL 应用架构与 13.2 节介绍的 MySQL 双主架构完全相同，所不同的是这里的双 Master 架构是通过 MMM 管理套件实现的，而前面介绍的双主架构是通过 Keepalived 实现的。

在通过 MMM 套件实现的双 Master 架构中，需要 5 个 IP 地址，两个 Master 节点各有一个固定不变的物理 IP 地址，另外还有两个只读 IP 和一个可写 IP，这三个虚拟 IP（2 个 reader IP 和一个 writer IP）不会固定在任何一个节点上，相反，它会在两个 Master 节点之间来回切换，如何切换取决于节点的可用性。例如，活动的 Master（Master1）节点发生故障，它的一个 reader IP 和一个 writer IP 将自动切换到备用 Master（Master2）节点上。

在正常情况下（系统、网络正常，MySQL 服务正常，主从复制正常，没有复制延迟等），

Master1 有两个虚拟 IP (reader IP 和 writer IP), Master2 有一个虚拟 IP (reader IP), 如果 Master1 发生故障, 那么所有的 reader 和 writer 虚拟 IP 都会被分配给 Master2。

在双 Master 节点的基础上, 增加多个 Slave 节点, 即可实现双主多从节点应用架构, 如图 13-13 所示。

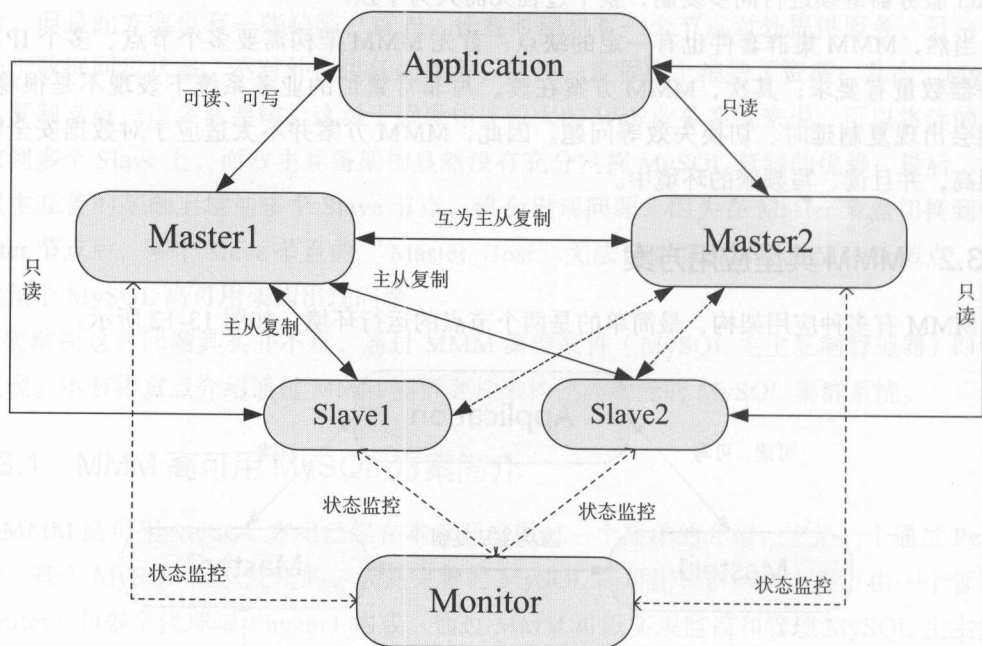


图 13-13 MMM 双主多从节点应用架构

双主多从节点的 MySQL 架构适合读查询量非常大的业务环境, 通过 MMM 提供的 reader IP 和 writer IP 可以轻松实现 MySQL 的读、写分离架构。此架构通过两个 Master 实现 MySQL 写操作的高可用, 然后在 Master 后端又增加了多个 Slave 节点, 所有的 Slave 节点只能进行读查询操作, 而多个 Slave 节点之间可以通过 LVS、HAProxy 等负载均衡软件实现 MySQL 读操作的负载均衡。

MMM 套件的优势在于: 它不但可以监控两个 Master 节点的运行状态, 还可以监控多个 Slave 节点的运行状态, 任何一个节点出现问题, 都会将失败节点对应的虚拟 IP 自动实现切换到其他健康的节点上, 保持读、写服务的连续性和高可用性。

MMM 不仅能提供虚拟 IP 自动转移功能, 更重要的是, 如果活动的 Master 节点发生故障, 会自动将后端的多个 Slave 节点转向备用的 Master 节点继续进行同步复制, 整个过程完全不需要手工更改同步复制的配置, 这是其他所有 MySQL 高可用集群方案都不具备的功能。

13.3.3 MMM 高可用 MySQL 方案架构

根据上面介绍的 MMM 应用方案，本小节将重点介绍通过 MMM 集群套件如何实现双主双从节点的典型应用架构，如图 13-14 所示。

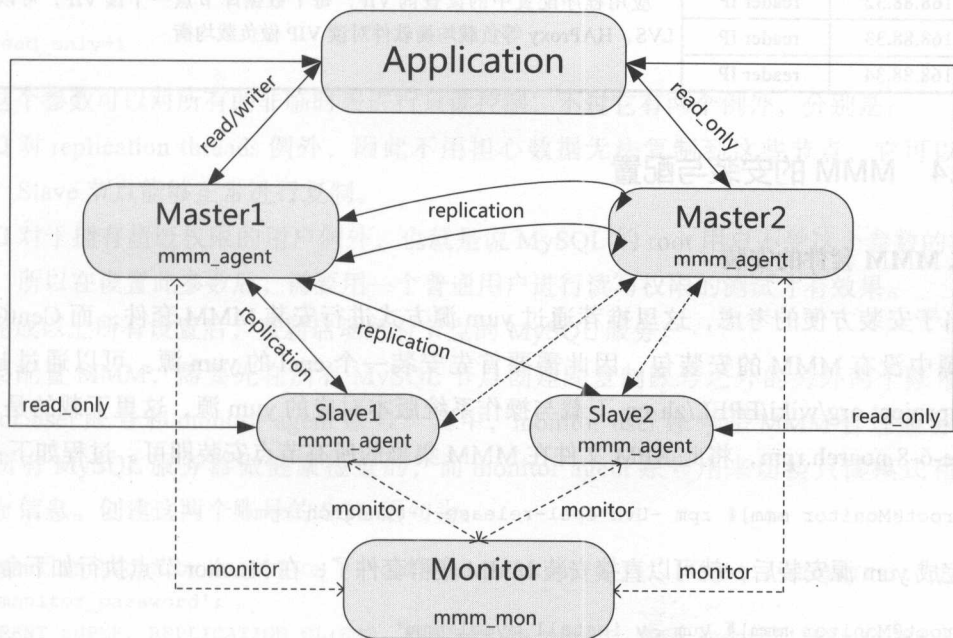


图 13-14 双主双从的 MySQL 高可用集群架构

此架构需要 5 台独立的服务器，其中一台 MMM 管理节点，两台 MySQL 的 Master 节点，还有两台 MySQL 的 Slave 节点，服务器配置环境如表 13.2 所示。

表 13-2 双主双从集群配置环境

主机名	系统版本	物理 IP	集群角色	MySQL 版本	server_id
Monitor	CentOS release 6.3	192.168.88.11	MMM 管理端	无	无
Master1	CentOS release 6.3	192.168.88.20	主 Master 可读、 可写	mysql-5.1.73	1
Master2	CentOS release 6.3	192.168.88.21	备 Master 可读、 可写	mysql-5.1.73	2
Slave1	CentOS release 6.3	192.168.88.22	Slave 节点只读	mysql-5.1.73	3
Slave2	CentOS release 6.3	192.168.88.23	Slave 节点只读	mysql-5.1.73	4

MMM 双主双从应用架构对应的读、写分离 IP 列表如表 13-3 所示。

表 13-3 双主双从应用架构读、写分离 IP 列表

虚拟 IP 地址	IP 角色	功能描述
192.168.88.30	writer IP	应用程序配置中的写入 VIP，仅支持单点写入
192.168.88.31	reader IP	应用程序配置中的读查询 VIP，每个数据库节点一个读 VIP，可以通过 LVS、HAProxy 等负载均衡软件对读 VIP 做负载均衡
192.168.88.32	reader IP	
192.168.88.33	reader IP	
192.168.88.34	reader IP	

13.3.4 MMM 的安装与配置

1. MMM 套件的安装

出于安装方便的考虑，这里推荐通过 yum 源方式进行安装 MMM 套件，而 CentOS 的默认源中没有 MMM 的安装包，因此需要首先安装一个 epel 的 yum 源。可以通过 <http://fedoraproject.org/wiki/EPEL/zh-cn> 下载与操作系统版本对应的 yum 源，这里下载的是 epel-release-6-8.noarch.rpm，将此 RPM 文件在 MMM 集群的所有节点安装即可，过程如下：

```
[root@Monitor mmm]# rpm -Uvh epel-release-6-8.noarch.rpm
```

完成 yum 源安装后，就可以直接安装 MMM 集群套件了，在 Mnoitor 节点执行如下命令：

```
[root@Monitor mmm]# yum -y install mysql-mmm*
```

在每个 MySQL DB 节点只需要安装 mysql-mmm-agent 即可，因此可执行如下命令：

```
[root@Master1 mysql]# yum -y install mysql-mmm-agent
```

完成安装后，查看下安装的 mysql-mmm 版本信息，执行如下命令：

```
[root@Monitor mmm]# rpm -qa|grep mysql-mmm
mysql-mmm-agent-2.2.1-2.el6.noarch
mysql-mmm-tools-2.2.1-2.el6.noarch
mysql-mmm-2.2.1-2.el6.noarch
mysql-mmm-monitor-2.2.1-2.el6.noarch
```

这里安装的 MMM 套件版本为 mysql-mmm-2.2.1-2。至此，MMM 集群套件安装完成。

2. MMM 集群套件的配置

在进行 MMM 套件配置之前，需要事先配置好 Master1 到 Master2 之间的主主互为同步，同时还要配置好 Master1 到 Slave1、Slave2 之间主从同步。关于 Master1 到 Master2 之间的主主同步，前面章节已经做过介绍，这里不再说明，而 Master1 到 Slave1、Slave2 之间主从同步也非常简单，不做过多介绍。但需要注意的是，在配置 Slave1、Slave2 和主 Master 同步时，


```

<host default>
    cluster_interface      eth0          # 配置的网络接口, 注意这里不能指定为子接口, 例如 eth0:0
    pid_path               /var/run/mysql-mmm/mmm_agentd.pid # 设定 PID 文件位置
    bin_path               /usr/libexec/mysql-mmm/          # 设定 MMM 可执行文件路径
    replication_user       repl_user      # 设置复制的用户名
    replication_password   repl_password  # 设置复制的密码
    agent_user             mmm_agent      # 设置更改只读操作的用户, 这个用户前面已经建立
    agent_password         agent_password # 设置更改只读操作用户的密码
</host>
<host db1>                # 设置 db1 的配置信息, db1 会在 mmm_agent.conf 文件中定义
    ip                    192.168.88.20  # 设置 db1 的物理 IP 地址
    mode                  master         # 设置 db1 的角色为 Master
    peer                  db2            # 设置与 db1 对等的主机名, 也就是 db1 和 db2 均为 Master 角色
</host>
<host db2>                # 设置 db2 的配置信息, db2 会在 mmm_agent.conf 文件中定义
    ip                    192.168.88.21  # 设置 db2 的物理 IP 地址
    mode                  master         # 设置 db2 的角色为 Master
    peer                  db1            # 设置与 db2 对等的主机名
</host>
<host db3>                # 设置 db3 的配置信息, db3 会在 mmm_agent.conf 文件中定义
    ip                    192.168.88.22  # 设置 db3 的物理 IP 地址
    mode                  slave          # 设置 db2 的角色为 Slave
</host>
<host db4>                # 设置 db4 的配置信息, db4 会在 mmm_agent.conf 文件中定义
    ip                    192.168.88.23
    mode                  slave
</host>
<role writer>             # 设置可写角色模式
    hosts                 db1, db2       # 设置可执行写操作的主机, 这里指定 db1 和 db2 均可写
    ips                   192.168.88.30  # 设置可写的虚拟 IP 地址
    mode                  exclusive      # 设置角色的模式为互斥, 互斥角色只有一个 IP, 并且同一时间
                                         # 只能分配给一个主机, 一般 writer 角色是 exclusive 模式
</role>
<role reader>             # 设置可读角色模式
    hosts                 db1, db2, db3, db4 # 设置可执行读操作的主机
    ips                   192.168.88.31, 192.168.88.32, 192.168.88.33, 192.168.88.34
                                         # 设置可读的虚拟 IP 地址, 可以有多个
    mode                  balanced        # 设置角色的模式为负载均衡, 在负载均衡角色中, 可以有多个 IP,
                                         # 这些 IP 被均衡动态地分配给多台 MySQL 主机, 一般 reader 角色
                                         # 是 balanced 模式
</role>

```

(2) 配置 mmm_agent.conf 文件

接下来看一下 mmm_agent.conf 文件, 此文件非常简单, 主要用来定义集群中的主机, 在 mmm_common.conf 文件中使用的 db1、db2 等名称都是在此文件中定义的。下面给出

Master1 节点 `mmm_agent.conf` 文件的内容：

```
include mmm_common.conf
this db1
```

其中，`include` 参数将 `mmm_common.conf` 文件引用了进来，`this` 参数指定了 Master1 节点在 MMM 集群中对应的主机名为 `db1`，其他节点 `mmm_agent.conf` 文件内容依此类推，即 Master2 节点对应的主机名为 `db2`，Slave1 节点对应的主机名为 `db3`，Slave2 节点对应的主机名为 `db4`。这里不再给出其他三个节点中 `mmm_agent.conf` 文件的内容。

(3) 配置 `mmm_mon.conf` 文件

`mmm_mon.conf` 配置文件仅在 MMM 管理节点上设置，配置好的 `mmm_mon.conf` 文件内容如下：

```
include mmm_common.conf
<monitor>
ip          127.0.0.1      # 为了安全性，设置只在本机监听，mmm_mond 默认监听端口为 9988
pid_path    /var/run/mysql-mmm/mmm_mond.pid    # 设置 mmm_mond 进程 PID 文件位置
bin_path    /usr/libexec/mysql-mmm             # MMM 可执行文件路径
status_path /var/lib/mysql-mmm/mmm_mond.status # MMM 集群的状态文件
ping_ips    192.168.88.1, 192.168.88.20, 192.168.88.21, 192.168.88.22,
192.168.88.23
# 用于测试网络可用性的 IP 地址列表，只要其中有一个地址能 ping 通，就代表网络正常，这里不要
# 写入本机的 IP 地址。
flap_duration 3600      # 抖动的时间范围，默认 3600s
flap_count    3         # 在抖动的时间范围内，最大的抖动次数
auto_set_online 0       # 是否设置自动上线，如果该值大于 0，抖动的主机在
                        # 抖动的时间范围过后，则设置自动上线
</monitor>

<host default>
monitor_user    mmm_monitor      # monitor user 账号，事先已经设置过
monitor_password monitor_password # monitor user 账号对应的密码
</host>

debug 0        # MMM 管理端的运行模式，1 为 debug 模式，0 为正常模式
```

(4) 配置 `mysql-mmm-agent` 文件

最后还有一个文件 `/etc/default/mysql-mmm-agent`，它要在 MMM 集群所有的 MySQL 节点进行设置，此文件的内容只有一行，确保此文件在所有 MySQL 节点的内容为：

```
ENABLED=1
```

至此，MMM 集群的 4 个主要配置文件介绍完毕。完成所有文件配置后，将 `mmm_common.conf` 文件从 MMM 集群管理节点依次复制到 4 个 MySQL 节点即可。这里需要注

意的是，MMM 集群中所有配置文件的权限最好都设置为 640，否则启动 MMM 服务的时候可能出错。

13.3.5 MMM 的管理

1. MMM 集群服务管理

MMM 集群套件在安装完成后会在系统的 /etc/init.d/ 目录下生成 mysql-mmm-monitor 和 mysql-mmm-agent 两个服务管理脚本，其中 mysql-mmm-monitor 文件用来管理 MMM 集群控制端的服务启动与关闭，而 mysql-mmm-agent 负责管理 MMM 集群中每个 agent 端（所有 MySQL 服务节点）服务的启动和关闭。这两个脚本的相关用法如下：

```
[root@monitor init.d]# /etc/init.d/mysql-mmm-monitor
Usage: /etc/init.d/mysql-mmm-monitor {start|stop|restart|condrestart|status}
```

```
[root@Master1 init.d]# /etc/init.d/mysql-mmm-agent
Usage: /etc/init.d/mysql-mmm-agent {start|stop|restart|condrestart|status}
```

在完成 MMM 集群配置后，就可以通过这两个脚本来启动 MMM 集群，首先在 MMM 集群管理端启动 mysql-mmm-monitor 服务，可执行如下命令：

```
[root@Monitor init.d]# /etc/init.d/mysql-mmm-monitor start
```

然后在每个 agent 端依次启动 agent 服务，执行操作如下：

```
[root@Master1 init.d]# /etc/init.d/mysql-mmm-agent start
```

这样 MMM 集群服务就可以运行了。

2. MMM 基本维护管理

查看集群的运行状态有两种方式，一种是通过 MMM 集群提供的 mmm_control（仅在 MMM 管理端存在）命令，另一种是查看 MMM 集群的运行日志信息，MMM 集群的运行日志位于每个集群节点的 /var/log/mysql-mmm 目录下，可通过查看日志文件了解集群的运行状态。这里重点介绍通过 mmm_control 工具查看和管理 MMM 集群。

要查看 mmm_control 工具的用法，执行“mmm_control help”命令即可显示，操作如下：

```
[root@Monitor init.d]# mmm_control help
Valid commands are:
```

help

显示帮助信息

ping

测试网络运行状态

show

显示 MMM 集群中所有节点的状态

checks [<host>|all [<check>|all]]

显示 MMM 集群中指定节点的详细状态或

显示所有节点的详细运行状态

```

set_online <host>          # 将一个 MMM 集群节点设置为 online 状态
set_offline <host>         # 将一个 MMM 集群节点设置为 offline 状态
mode                       # 显示 MMM 集群当前的运行模式，有 active、manual 和
                           # passive 三种模式，默认是 active 模式
set_active                 # 切换 MMM 集群到 active 模式
set_manual                 # 切换 MMM 集群到 manual 模式
set_passive                # 切换 MMM 集群到 passive 模式
move_role [--force] <role> <host> # 在互斥模式下切换角色，例如将写操作
                                # 从 db1 切换到 db2
set_ip <ip> <host>        # 用来在被动模式下操纵角色

```

下面列举几个常用的 MMM 集群维护管理的例子。例如，要查看集群运行状态，可执行如下命令：

```

[root@Monitor mysql-mmm]# mmm_control show
db1(192.168.88.20) master/ONLINE. Roles: reader(192.168.88.31), writer(192.168.88.30)
db2(192.168.88.21) master/ONLINE. Roles: reader(192.168.88.32)
db3(192.168.88.22) slave/ONLINE. Roles: reader(192.168.88.33)
db4(192.168.88.23) slave/ONLINE. Roles: reader(192.168.88.34)

```

在这个输出中，最左边的 IP 信息是集群节点的物理 IP，然后是 MySQL 主从复制的角色和集群节点的状态，最后是 MMM 集群中读、写角色以及对应的读、写虚拟 IP 地址信息。这里需要注意的是 MySQL 主从复制的角色和集群节点的状态，例如 master/ONLINE 表示主从复制的 Master 角色处于正常的在线状态。在 MMM 集群中，集群节点的状态有如下几种：

- ONLINE：表示节点运行正常，处于在线状态。
- ADMIN_OFFLINE：表示节点是通过手动模式离线的。
- HARD_OFFLINE：表示节点处于离线状态，这一般是由于 MMM 集群脚本执行 ping 操作失败或检测 MySQL 失败而切换的一种状态。
- AWAITING_RECOVERY：表示等待恢复状态，如果 MMM 集群设置的是 active 运行模式，那么此状态将会自动恢复到 ONLINE 状态。
- REPLICATION_FAIL：表示主从复制失败状态，一般是由于复制主线程没有运行导致的。
- REPLICATION_DELAY：表示复制日志有延时，一般是由于检查日志失败导致的。

如果要查看 MMM 集群目前处于什么运行模式，可执行如下命令：

```

[root@Monitor mysql-mmm]# mmm_control mode
ACTIVE

```

由输出可知，MMM 集群目前运行于 active 模式，MMM 支持 active、manual 和 passive 三种模式。

□ **active**：表示主动模式，该模式是默认的，即活动的 Master 发生故障后另一个备份的 Master 可自动接管故障 Master 的角色，继续提供服务；如果 Slave 节点发生故障，其他健康的 Slave 节点也会自动接管故障 Slave 节点的服务，继续提供服务。此模式是经常使用的。

□ **manual**：表示手动模式，该模式不会执行自动切换操作，如果某个集群节点发生故障，需要手动切换到其他健康的节点。此模式一般用于特殊的场景中。

□ **passive**：表示被动模式，在被动模式下，MMM 管理端不会改变集群中节点的角色，也不更新状态文件和发送任何信息给每个 agent 节点，在启动的节点角色发生冲突时将会进入被动模式。此模式一般不用。

如果要查看所有 MMM 集群节点的运行状态，可执行如下命令：

```
[root@Monitor mysql-mmm]# mmm_control checks all

db4 ping [last change: 2014/03/30 13:46:53] OK
db4 mysql [last change: 2014/03/30 13:46:53] OK
db4 rep_threads [last change: 2014/03/30 14:49:10] OK
db4 rep_backlog [last change: 2014/03/30 13:46:53] OK: Backlog is null
db2 ping [last change: 2014/03/30 13:46:53] OK
db2 mysql [last change: 2014/03/30 13:48:23] OK
db2 rep_threads [last change: 2014/03/30 15:16:30] OK
db2 rep_backlog [last change: 2014/03/30 13:46:53] OK: Backlog is null
db3 ping [last change: 2014/03/30 13:46:53] OK
db3 mysql [last change: 2014/03/30 13:46:53] OK
db3 rep_threads [last change: 2014/03/30 14:49:10] OK
db3 rep_backlog [last change: 2014/03/30 13:46:53] OK: Backlog is null
db1 ping [last change: 2014/03/30 13:46:53] OK
db1 mysql [last change: 2014/03/30 15:15:49] OK
db1 rep_threads [last change: 2014/03/30 13:49:04] OK
db1 rep_backlog [last change: 2014/03/30 13:47:36] OK: Backlog is null
```

而要单独查看某个节点的运行状态，可执行如下命令：

```
[root@Monitor mysql-mmm]# mmm_control checks db1

db1 ping [last change: 2014/03/30 13:46:53] OK
db1 mysql [last change: 2014/03/30 15:15:49] OK
db1 rep_threads [last change: 2014/03/30 13:49:04] OK
db1 rep_backlog [last change: 2014/03/30 13:47:36] OK: Backlog is null
```

从这个输出可以看出，mmm_mond 进程会对每个节点执行四项检查，并且确定检查是否成功，这四项检查的含义为：

□ **ping** 主要用来测试网络可用性。

□ **mysql** 用来检测 MySQL 服务器是否运行正常。

□ rep_threads 用来检测 MySQL 的复制线程是否正常运行。

□ rep_backlog 用来检测 MySQL 的复制日志是否有挤压。

在这四项检测中,任何一项出现问题,都会进行角色切换操作,MMM 集群正是通过这四项健康检查来保证 MySQL 服务的高可用性。

13.3.6 测试 MMM 实现 MySQL 高可用功能

为了确保 MMM 集群已经正常运行,下面重点测试一下 MMM 所实现的读、写分离功能和故障转移功能。

1. 读、写分离测试

要在任意一个 MySQL 远程客户端通过 MMM 提供的读、写虚拟 IP 地址登录 MMM 集群,测试一下是否实现了数据同步和读、写限制,首先使用 MMM 集群的可写 VIP 远程登录,操作过程如图 13-15 所示。

```
[root@apps ~]# mysql -uixdba -p -h 192.168.88.30
mysql> show variables like "%hostname%";
+-----+-----+
| Variable_name | Value |
+-----+-----+
| hostname      | Master1 |
+-----+-----+
1 row in set (0.01 sec)

mysql> use repldb;
mysql> create table mmm_test(id int, email varchar(30));
Query OK, 0 rows affected (0.02 sec)

mysql> insert into mmm_test (id,email) values(189,"mmm_mysql@189.cn");
Query OK, 1 row affected (0.02 sec)

mysql> select * from mmm_test;
+----+-----+
| id | email |
+----+-----+
| 189 | mmm_mysql@189.cn |
+----+-----+
1 row in set (0.00 sec)
```

图 13-15 通过写 VIP 测试 MySQL 写操作

由上面的操作可知,通过可写的 VIP 登录了 Master1 节点,这里创建了一张表 mmm_test,并且插入了一条数据。此时可以登录 Master2 节点、Slave1 节点和 Slave2 节点,查看下数据是否已经同步过去。

接着仍在 MySQL 远程客户端通过 MMM 提供的只读 VIP 登录 MySQL 集群,测试过程如图 13-16 所示。


```
[root@webapp ~]# mysql -uixdba -p -h 192.168.88.32
mysql> show variables like "%hostname%";
+-----+-----+
| Variable_name | Value |
+-----+-----+
| hostname      | Slave1 |
+-----+-----+
1 row in set (0.03 sec)
mysql> use repldb;
mysql> select * from mmm_test;
+-----+-----+
| id | email |
+-----+-----+
| 189 | mmm_mysql@189.cn |
+-----+-----+
1 row in set (0.02 sec)
mysql> create table mmm_test1(id int,email varchar(100));
ERROR 1290 (HY000):
The MySQL server is running with the --read-only option so it cannot execute this statement
```

图 13-16 通过读 VIP 测试数据同步和读、写限制功能

在这个操作过程中，通过只读虚拟 IP 地址“192.168.88.32”登录了 Slave1 主机，并且在 Master1 节点创建的表 mmm_test 已经同步过来。最后尝试在 Slave1 上创建表 mmm_test1 时，发生错误，提示 Slave1 节点设置了只读模式。这刚好实现了读、写分离的限制，由此可知，MMM 集群不但实现了数据的同步，而且通过可写 VIP 及只读 VIP 实现了读、写分离的功能。

2. 故障转移功能测试

故障转移分为 Master 节点故障转移和 Slave 节点故障转移。为了进行状态比较，这里首先检查下 MMM 目前的集群运行状态，信息如下：

```
[root@Monitor mysql-mmm]# mmm_control show
db1(192.168.88.20) master/ONLINE. Roles: reader(192.168.88.31), writer(192.168.88.30)
db2(192.168.88.21) master/ONLINE. Roles: reader(192.168.88.32)
db3(192.168.88.22) slave/ONLINE. Roles: reader(192.168.88.33)
db4(192.168.88.23) slave/ONLINE. Roles: reader(192.168.88.34)
```

接着，关闭 Master1 节点的 MySQL 服务，再次查看 MMM 集群运行状态，信息如下：

```
[root@Monitor mysql-mmm]# mmm_control show
db1(192.168.88.20) master/HARD_OFFLINE. Roles:
db2(192.168.88.21) master/ONLINE. Roles: reader(192.168.88.32), writer(192.168.88.30)
db3(192.168.88.22) slave/ONLINE. Roles: reader(192.168.88.31), reader(192.168.88.33)
db4(192.168.88.23) slave/ONLINE. Roles: reader(192.168.88.34)
```

从输出信息可知，Master1 目前处于 HARD_OFFLINE 状态，之前在 Master1 上面运行的写 VIP 自动切换到 Master2，而读 VIP 自动切换到 Slave1。为了验证切换后是否正常，还可以继续执行上面的读、写分离测试，检查一下 Master 节点切换后，能否正常实现数据

的同步功能以及读、写分离的限制。这个测试过程这里不再重复讲述了。

接着上面的测试继续进行，重新启动 Master1 节点上的 MySQL 服务，然后查看 MMM 集群的运行状态，信息如下：

```
[root@Monitor mysql-mmm]# mmm_control show
db1(192.168.88.20) master/ONLINE. Roles: reader(192.168.88.31)
db2(192.168.88.21) master/ONLINE. Roles: reader(192.168.88.32), writer(192.168.88.30)
db3(192.168.88.22) slave/ONLINE. Roles: reader(192.168.88.33)
db4(192.168.88.23) slave/ONLINE. Roles: reader(192.168.88.34)
```

从输出可知，在 Master1 恢复正常后，读 VIP 地址“192.168.88.31”又自动切换到 Master1 节点，而写 VIP 地址“192.168.88.30”却一直留在 Master2 节点。这种机制类似于 Keepalived 中的不抢占功能，由于 MySQL 故障的切换代价很大，所以即使 Master1 恢复正常，写 VIP 地址也不再切换回来，直到 Master2 发生故障时才会再次进行切换。而如果要让写 VIP 地址在 Master1 节点运行，可以手动执行如下切换操作：

```
[root@Monitor mysql-mmm]# mmm_control move_role writer db1
OK: Role 'writer' has been moved from 'db2' to 'db1'. Now you can wait some time
and check new roles info!
[root@Monitor mysql-mmm]# mmm_control show
db1(192.168.88.20) master/ONLINE. Roles: reader(192.168.88.31), writer(192.168.88.30)
db2(192.168.88.21) master/ONLINE. Roles: reader(192.168.88.32)
db3(192.168.88.22) slave/ONLINE. Roles: reader(192.168.88.33)
db4(192.168.88.23) slave/ONLINE. Roles: reader(192.168.88.34)
```

这样，MMM 集群又恢复了初始状态。

最后，再测试一下 MMM 集群对 Slave 节点的 MySQL 复制线程的监控和故障转移过程。首先在 Slave1 节点关闭复制线程，操作如下：

```
[root@Slave1 ~]# mysql -uroot -p
Enter password:
mysql> slave stop;
Query OK, 0 rows affected (0.02 sec)
```

接着在 MMM 管理端查看 MMM 集群运行状态，信息如下：

```
[root@Monitor mysql-mmm]# mmm_control show
db1(192.168.88.20) master/ONLINE. Roles: reader(192.168.88.31), writer(192.168.88.30)
db2(192.168.88.21) master/ONLINE. Roles: reader(192.168.88.33), reader(192.168.88.32)
db3(192.168.88.22) slave/REPLICATION_FAIL. Roles:
db4(192.168.88.23) slave/ONLINE. Roles: reader(192.168.88.34)
```

从输出可知，Slave1 的状态变为 REPLICATION_FAIL，同时只读 VIP 切换到 Master2 节点，实现故障的快速转移。如果重新启动 Slave1 上的复制线程，那么 MMM 集群也会立刻检测到，进而再次将只读 VIP 切换到 Slave1 上。

13.4 MySQL 读写分离解决方案

在 MMM 集群架构中, 通过提供虚拟的读、写 IP 地址, 将数据库的读写功能分离出来, 但是这仅仅设定了读、写的 VIP 地址, 并没有真正实现业务系统中所说的读、写分离功能, 因为应用程序不可能在需要读的时候就去找可读的 VIP, 在写的时候就去找可写的 VIP, 要解决这个问题, 可由两种方法来实现。

第一种实现读、写分离的方式是通过修改程序, 将读、写操作提取出来, 并分别在程序的连接池中设定可读、可写的 VIP 地址, 这种方法要修改业务系统的程序, 实现起来相对比较困难, 如果是新开发的程序, 可以在开发时就预留这样的接口, 而如果程序已经在运行, 修改的难度是相当大的, 采用这种方法基本行不通。

第二种实现读、写分离的方法是通过一个数据库透明代理, 也就是在业务系统和数据库之间提供一个代理接口, 由这个接口来完成业务系统读、写请求的分发, 将读操作分发到后端只读的数据库服务器上, 而将写请求分发到后端可写的数据库服务器上。常见的读、写分离软件有 Amoeba 和 MySQL-Proxy。

MySQL-Proxy 是 MySQL 官方推出的一个处在业务系统和 MySQL 数据库之间的程序, 这个代理可以用来分析、监控和变换通信数据, 但是 MySQL 官方建议不要将 MySQL-Proxy 用于生产环境。事实上, MySQL-Proxy 确实很不稳定, 它的读、写分离功能都是通过一个 lua 脚本来实现的, 而这个脚本 bug 很多, 所以不建议通过 MySQL-Proxy 来实现读、写分离功能, 不过可以作为线下测试使用。

Amoeba 是一个开源项目, 致力于 MySQL 的分布式数据库前端代理层, 它主要在应用层访问 MySQL 的时候充当 SQL 路由器功能, 具有负载均衡、高可用性、SQL 过滤、读写分离等功能, 通过 Amoeba 可以实现数据源的高可用、负载均衡、数据切片等功能。本节将重点介绍 Amoeba 作为 MySQL 读、写分离代理接口的实现过程。

13.4.1 通过 Amoeba 实现 MySQL 读写分离

1. MMM 整合 Amoeba 应用架构

在实际的应用环境中, Amoeba 可与简单的 MySQL 主从复制架构进行整合, 实现读与写的分离操作。但是这样存在安全性问题, 例如 MySQL 的 Master 节点出现故障或者任何一个 Slave 节点故障, 那么 Amoeba 并不能自动屏蔽这些故障的 MySQL 节点, 可能会导致前端应用程序无法读取数据库的情况。为了解决这个问题, Amoeba 经常与 MMM 集群架构一起使用, 这样如果任意 MySQL 节点故障, MMM 集群就能自动屏蔽故障节点, 从而保证 Amoeba 一直能够连接到正常的 MySQL 节点。这里要介绍的 Amoeba 应用环境就是在


```
lib/tools.jar
export PATH=$JAVA_HOME/bin:$PATH
```

将这些内容添加到系统的 `/etc/profile` 文件中即可完成 Java 环境的设置。

3. 配置 Amoeba

Amoeba 的配置文件在本环境下位于 `/usr/local/amoeba/conf` 目录。配置文件比较多,但是仅仅使用读、写分离功能,只需配置二个文件即可,分别是 `dbServers.xml` 和 `amoeba.xml`,如果需要配置 IP 访问控制,还需要修改 `access_list.conf` 文件。下面首先介绍 `dbServers.xml` 文件的配置,内容如下:

```
<?xml version="1.0" encoding="gbk"?>
<!DOCTYPE amoeba:dbServers SYSTEM "dbserver.dtd">
<amoeba:dbServers xmlns:amoeba="http://amoeba.meidusa.com/">
  <dbServer name="abstractServer" abstractive="true">
    <factoryConfig class="com.meidusa.amoeba.mysql.net.MysqlServerConnectionFactory">
      <property name="connectionManager">${defaultManager}</property>
      <property name="sendBufferSize">64</property>
      <property name="receiveBufferSize">128</property>
      # 下面这个配置是设置 Amoeba 要连接的 mysql 数据库的端口, 默认是 3306
      <property name="port">3306</property>
      # 下面这个配置是设置 Amoeba 默认连接的数据库名, 当连接业务系统连接 amoeba 时, 操作
      # 表必须显式指定数据库名, 即采用 dbname.tablename 的方式, 不支持 "use dbname"
      # 指定默认库
      <property name="schema">repldb</property>
      # 下面这两个配置是设置 Amoeba 连接后端数据库服务器的账号和密码, 因此需要在所有后
      # 端数据库服务器上创建该用户, 并授权 Amoeba 服务器可连接
      <property name="user">ixdba</property>
      <property name="password">xxxxxx</property>
    </factoryConfig>
    <poolConfig class="com.meidusa.toolkit.common.poolable.PoolableObjectPool">
      <property name="maxActive">500</property> # 配置最大连接数, 默认是 500
      <property name="maxIdle">500</property> # 配置最大空闲连接数
      <property name="minIdle">1</property> # 配置最小空闲连接数
      <property name="minEvictableIdleTimeMillis">600000</property>
      <property name="timeBetweenEvictionRunsMillis">600000</property>
      <property name="testOnBorrow">true</property>
      <property name="testOnReturn">true</property>
      <property name="testWhileIdle">true</property>
    </poolConfig>
  </dbServer>
  # 下面这个配置用来设置一个后端可写 dbServer, 这里定义为 writedb, 这个名字可以任意命名, 后面还
  # 会用到
  <dbServer name="writedb" parent="abstractServer">
    <factoryConfig>
```

下面这个配置是指定可写数据库的 IP 地址, 也就是 MMM 集群提供对外访问的可写 VIP 地址

```
<property name="ipAddress">192.168.88.30</property>
</factoryConfig>
</dbServer>
```

下面这段配置用来设置一个后端可读 dbServer, 这里定义了 4 台可读 dbServer, 分别是 slave1、slave2、slave3 和 slave4, 这个名字也可以任意命名, 每个 dbServer 对应的 IP 地址就是 MMM 集群提供对外访问的可读 VIP 地址

```
<dbServer name="slave1" parent="abstractServer">
  <factoryConfig>
    <property name="ipAddress">192.168.88.31</property>
  </factoryConfig>
</dbServer>
<dbServer name="slave2" parent="abstractServer">
  <factoryConfig>
    <property name="ipAddress">192.168.88.32</property>
  </factoryConfig>
</dbServer>
<dbServer name="slave3" parent="abstractServer">
  <factoryConfig>
    <property name="ipAddress">192.168.88.33</property>
  </factoryConfig>
</dbServer>
<dbServer name="slave4" parent="abstractServer">
  <factoryConfig>
    <property name="ipAddress">192.168.88.34</property>
  </factoryConfig>
</dbServer>
```

下面的配置是定义一个虚拟的 dbServer, 实际上相当于一个 dbServer 组, 这里将可读的数据库 IP 统一放到一个组中, 将这个组的名字命名为 myslaves

```
<dbServer name="myslaves" virtual="true">
  <poolConfig class="com.meidusa.amoeba.server.MultipleServerPool">
```

下面的配置是选择调度算法 1 表示复制均衡, 2 表示权重, 3 表示 HA, 这里选择 1, 实现读操作的负载均衡

```
  <property name="loadbalance">1</property>
  # 下面的配置是设置 dbServer 组中的成员, 这里将所有可读的 dbServer 全部加进来
  <property name="poolNames">slave1,slave2,slave3,slave4</property>
</poolConfig>
```

```
</dbServer>
</amoeba:dbServers>
```

另一个配置文件 amoeba.xml 的内容如下:

```
<?xml version="1.0" encoding="gbk"?>
<!DOCTYPE amoeba:configuration SYSTEM "amoeba.dtd">
<amoeba:configuration xmlns:amoeba="http://amoeba.meidusa.com/">
  <proxy>
    <service name="Amoeba for Mysql" class="com.meidusa.amoeba.mysql.server.MySQLService">
```

下面的配置是设置 Amoeba 监听的端口，默认是 8066，可根据情况进行修改

```
<property name="port">8066</property>
```

下面的配置是设置监听的接口，如果不设置，默认监听所有的 IP

```
<!--
```

```
<property name="ipAddress">127.0.0.1</property>
```

```
-->
```

```
<property name="connectionFactory">
```

```
<bean class="com.meidusa.amoeba.mysql.net.MysqlClientConnectionFactory">
```

```
<property name="sendBufferSize">128</property>
```

```
<property name="receiveBufferSize">64</property>
```

```
</bean>
```

```
</property>
```

```
<property name="authenticateProvider">
```

```
<bean class="com.meidusa.amoeba.mysql.server.MysqlClientAuthenticator">
```

下面这两个配置主要是设置客户端连接 Amoeba 时需要使用的账号和密码，需要说明的是，如果部署了 Amoeba，
那么业务系统中程序连接数据库的地址就是 #Amoeba 服务器的地址，连接用户名和密码就是这里设置的
账号和密码

```
<property name="user">root</property>
```

```
<property name="password">xxxxxx</property>
```

```
<property name="filter">
```

```
<bean class="com.meidusa.toolkit.net.authenticate.server.IPAccessController">
```

```
<property name="ipFile">${amoeba.home}/conf/access_list.conf</property>
```

```
</bean>
```

```
</property>
```

```
</bean>
```

```
</property>
```

```
</service>
```

```
<runtime class="com.meidusa.amoeba.mysql.context.MysqlRuntimeContext">
```

```
<property name="executeThreadSize">128</property>
```

```
<property name="statementCacheSize">500</property>
```

```
<property name="serverCharset">utf8</property>
```

```
<property name="queryTimeout">60</property>
```

```
</runtime>
```

```
</proxy>
```

```
<connectionManagerList>
```

```
<connectionManager name="defaultManager" class="com.meidusa.toolkit.net.  
MultiConnectionManagerWrapper">
```

```
<property name="subManagerClassName">com.meidusa.toolkit.net.
```

```
AuthingableConnectionManager</property>
```

```
</connectionManager>
```

```
</connectionManagerList>
```

```
<dbServerLoader class="com.meidusa.amoeba.context.DBServerConfigFileLoader">
```

```
<property name="configFile">${amoeba.home}/conf/dbServers.xml</property>
```

```
</dbServerLoader>
```

```
<queryRouter class="com.meidusa.amoeba.mysql.parser.MysqlQueryRouter">
```

```

<property name="ruleLoader">
    <bean class="com.meidusa.amoeba.route.TableRuleFileLoader">
        <property name="ruleFile">${amoeba.home}/conf/rule.xml</property>
        <property name="functionFile">${amoeba.home}/conf/ruleFunctionMap.xml</property>
    </bean>
</property>
<property name="sqlFunctionFile">${amoeba.home}/conf/functionMap.xml</property>
<property name="LRUMapSize">1500</property>
    # 下面这个选项设置 Amoeba 默认的池, 这里设置为 writedb
<property name="defaultPool">writedb</property>
    # 下面这两个选项默认是注释掉的, 需要取消注释, 这里用来指定前面定义好的两个读、写池,
    # 分别是可写池 writedb 和可读池 slaves
    <property name="writePool">writedb</property>
    <property name="readPool">myslaves</property>
    <property name="needParse">true</property>
</queryRouter>
</amoeba:configuration>

```

这样 Amoeba 就配置完成了。

4. 设置 Amoeba 登录数据库权限

这里假定 Amoeba 服务器的 IP 地址为 192.168.88.35, 在 MMM 集群的所有 MySQL 节点上执行如下操作, 为 Amoeba 访问 MMM 集群中所有 MySQL 数据库节点授权:

```

mysql> GRANT ALL ON repldb.* TO 'ixdba'@'192.168.88.35' IDENTIFIED BY 'xxxxxx';
mysql> flush privileges;

```

5. 启动 Amoeba

在 Amoeba 服务器上执行如下命令, 启动 Amoeba:

```

[root@amoebaserver bin]# /usr/local/amoeba/bin/launcher
2014-03-18 18:17:45 [INFO] Project Name=Amoeba-MySQL, PID=22474 , starting...
log4j:WARN log4j config load completed from file:/usr/local/amoeba/conf/log4j.xml
2014-03-18 18:17:50,462 INFO context.MysqlRuntimeContext - Amoeba for Mysql
current versoin=5.1.45-mysql-amoeba-proxy-3.0.4-BETA
log4j:WARN ip access config load completed from file:/usr/local/amoeba/conf/access_
list.conf
2014-03-18 18:17:55,643 INFO net.ServerableConnectionManager - Server listening
on 0.0.0.0/0.0.0.0:8066.
[root@cloud0 bin]# netstat -tulnp |grep java
tcp        0      0 :::8066 :::*   LISTEN    22474/java

```

由此可知 Amoeba 启动正常。

6. 测试 Amoeba 实现读、写分离和负载均衡

要测试 Amoeba 实现读、写分离和负载均衡功能, 需要在 MMM 集群的所有 MySQL 节

点开启 MySQL 的查询日志。查询日志可以记录数据库中建立的客户端连接和执行的语句。开启方法很简单，在 MySQL 配置文件 `/etc/my.cnf` 中添加如下内容：

```
log=/var/log/mysql_query_log
```

当然，`mysql_query_log` 文件要事先存在，并且对 MySQL 用户可写。

为了测试方便，这里在每个 MySQL 节点的 `test` 库中创建一张表，表的名字为 `mmm_test`，例如在 Master1 节点，创建表的过程如下：

```
mysql> use test;
mysql> create table mmm_test(id int,email varchar(60)) ;
mysql> insert into mmm_test (id,email) values (100,'this is 192.168.88.20');
```

上面的操作是在 `mmm_test` 表的 `email` 字段中插入一条记录 “this is 192.168.88.20”，而字符串 “this is 192.168.88.20” 就是个 IP 标识，这样做的目的是区分多个 MySQL 节点不同 IP 地址的情况。接着在 Master2 节点同样执行上面的 SQL 操作，所不同的是 `mmm_test` 表 `email` 字段的内容修改为 “this is 192.168.88.21”。依此类推，分别在 Slave1 和 Slave2 节点执行相同的操作。

接着，在远程 MySQL 客户端通过 Amoeba 配置文件中指定的用户名、密码、端口以及 Amoeba 服务器的 IP 地址连接 MySQL 数据库，操作过程如图 13-18 所示。

```
[root@webserver ~]# mysql -uroot -p -h 192.168.88.35 -P8066
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 357451187
Server version: 5.1.45-mysql-amoeba-proxy-3.0.4-BETA Source distribution
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql> select * from test.mmm_test;
+-----+-----+
| id | email |
+-----+-----+
| 100 | this is 192.168.88.22 |
+-----+-----+
1 row in set (0.06 sec)

mysql> select * from test.mmm_test;
+-----+-----+
| id | email |
+-----+-----+
| 100 | this is 192.168.88.21 |
+-----+-----+
1 row in set (0.08 sec)

mysql> select * from test.mmm_test;
+-----+-----+
| id | email |
+-----+-----+
| 100 | this is 192.168.88.20 |
+-----+-----+
1 row in set (0.01 sec)

mysql> select * from test.mmm_test;
+-----+-----+
| id | email |
+-----+-----+
| 100 | this is 192.168.88.23 |
+-----+-----+
1 row in set (0.02 sec)
```

图 13-18 测试 Amoeba 实现读操作的负载均衡功能

从图 13-18 可以看出，客户端连接到的 Server version 为 5.1.45-mysql-amoeba-proxy-3.0.4-BETA，可见客户端连接的是 Amoeba 实例而不是 MySQL 实例，而从下面的查询 test 库中 mmm_test 表的内容来看，Amoeba 依次将 4 次 select 请求均衡地分配到 MMM 集群中 4 个可读 MySQL 节点上，由此可知，Amoeba 实现了读操作的负载均衡。

下面继续进行 SQL 测试，创建两个表 mmm_test1 和 mmm_test2，操作过程如下：

```
mysql> create table mmm_test1(id int,email varchar(60));
Query OK, 0 rows affected (0.04 sec)

mysql> create table mmm_test2(id int,email varchar(60));
Query OK, 0 rows affected (0.04 sec)

mysql> insert into mmm_test1 (id,email) values (101,'mmm_test1@126.com');
Query OK, 1 row affected (0.02 sec)

mysql> drop table mmm_test2;
Query OK, 0 rows affected (0.10 sec)
```

为了确定创建的两个表是否已经正常同步到 MMM 集群的其他节点，可分别登录每个 MySQL 节点进行查询，接着还要确定 MySQL 的写操作是否分配到可写的节点 Master1，可通过查看每个 MySQL 节点的查询日志。Master1 节点的 MySQL 查询日志信息如图 13-19 所示。

```
[root@Master1 ~]# tail -f /var/log/mysql_query_log |grep mmm_test
36361 Query      select * from test.mmm_test
140331 16:42:05 36400 Query      create table mmm_test1(id int,email varchar(60))
36400 Query      create table mmm_test2(id int,email varchar(60))
36400 Query      insert into mmm_test1 (id,email) values (101,'mmm_test1@126.com')
36400 Query      drop table mmm_test2
```

图 13-19 Master1 节点 MySQL 查询日志

Master2 节点的 MySQL 查询日志信息如图 13-20 所示。

```
[root@Master2 ~]# tail -f /var/log/mysql_query_log |grep mmm_test
140331 16:41:48 39077 Query      select * from test.mmm_test
1 Query
140331 16:42:10 1 Query      create table mmm_test1(id int,email varchar(60))
140331 17:20:49 1 Query      create table mmm_test2(id int,email varchar(60))
140331 17:21:21 1 Query      insert into mmm_test1 (id,email) values (101,'mmm_test1@126.com')
1 Query      drop table mmm_test2
```

图 13-20 Master2 节点 MySQL 查询日志

其他节点的信息基本类似，这里不一一列出。从 MySQL 的查询日志中可以进一步验证 Amoeba 实现了读负载均衡，而写操作在 Master1 节点执行了。虽然 Master2 节点也有相关写操作的日志，但这是 MySQL 的复制线程执行的写操作，因为除了 Master1，其他 MySQL 节点都是 read_only 状态，是无法执行写操作的。

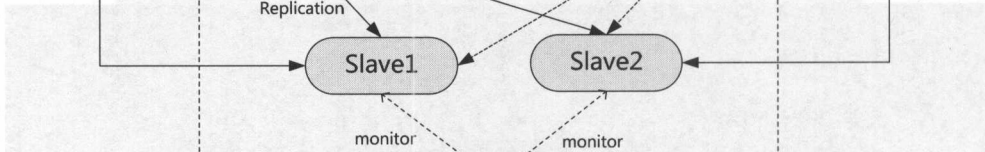


图 13-21 通过 Keepalived 实现 Amoeba 高可用并实现 MySQL 集群读、写分离架构

至此，关于如何构建高性能的 MySQL 集群系统介绍完毕。

高性能负载均衡集群软件 HAProxy

14.1 高性能负载均衡架构设计原则

对于一个负载均衡集群系统来说，高性能体现在高可用性、可扩展性、可视可控性和安全性这几个方面，每个方面要实现的功能如下：

- 高可用性：这是衡量一个应用系统最基础的标准，在生产环境中使用的负载均衡集群系统都要求有很强的实时性和可靠性，必须保证服务 24 小时不间断运行，而由于软件、硬件、网络、人为等各种原因，单一的服务运行环境很难达到这种要求，此时就需要有一个高可用的软件来保证负载均衡系统持续不间断地服务。高可用软件最大的一个优点是能保证负载均衡系统的高可用性，在负载均衡服务出现故障时，高可用软件可以自动将服务从故障节点切换到另一个备用节点，从而提供不间断性服务，保证了业务的持续运行。
- 可扩展性：随着业务量的加大，现有的集群服务实体不能满足需求时，可以向此集群中动态地加入一个或多个服务节点，从而满足应用的需要，增强集群的整体性能，这就是集群的可扩展性。
- 可视可控性：集群的服务状态、服务节点运行状态都应该能在一个统一的监控平台下实时监控，并且当集群服务或服务节点出现故障时，能够通过电话、短信、邮件等方式第一时间通知运维人员。
- 安全性：集群的整个运行架构应该是合理并安全的。安全防护包括两个层面，第一个层面就是服务器健康检查功能，负载均衡器要能够精确探测后端服务器的健康状

态，在后端服务器处理能力达到饱和前可以自动屏蔽新来的连接请求，以免后端服务器可能在瞬间接受超过服务器吞吐能力的数据流而直接导致系统崩溃，从而达到安全检测的目的。第二个层面是负载均衡器上要有相应的安全机制，例如端口屏蔽、ARP 响应屏蔽等策略，以保证负载均衡器不受外界的攻击。

高性能的负载均衡系统建立在现有网络结构之上，它提供了一种廉价有效的方法扩展服务器带宽和增加吞吐量，加强网络数据处理能力，提高了网络的灵活性和可用性。

14.1.1 HAProxy 常见方案与拓扑

HAProxy 是一个自由软件，任何人都可以免费获取并使用，而且 Linux 也是一个开源操作系统，这二者的组合大大节约了企业的应用成本；同时 HAProxy 具有高稳定性和高可靠性，在高并发和高吞吐量情况下，具有高负荷处理能力；当某个服务节点出现故障时，并不影响整个系统服务的正常运行。这些优点使 HAProxy 已经广泛应用在企业、教育行业以及很多知名网站。

目前业界常用的 HAProxy 构建方案如图 14-1 所示。

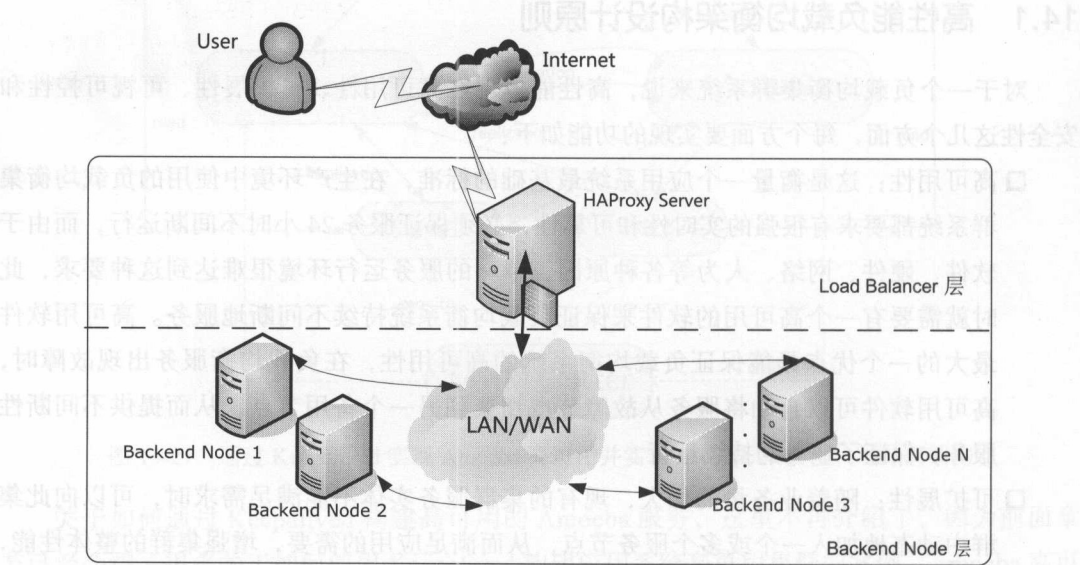


图 14-1 单节点的 HAProxy 负载均衡集群应用方案

在这个负载均衡集群架构中，每个模块实现功能如下：

- ❑ Load Balancer 层：位于整个集群系统的最前端，由一台 HAProxy Server 组成。HAProxy 软件就安装在 HAProxy Server 上，而 HAProxy Server 的主要作用类似于一个代理路由器，它把用户的请求分发给 Backend Node 层的后端服务器（Backend

Node)。同时,在 HAProxy Server 上还要配置对 Backend Node 进行状态监控的设置,通过对 Backend Node 服务的健康状况监控,以确保在 Backend Node 不可用时可以把它从整个集群中剔除,在恢复时重新加入。

□ Backend Node 层:由一组实际运行应用服务的机器组成。Backend Node 可以是 Web 服务器、Mail 服务器、FTP 服务器、DNS 服务器、视频服务器中的一个或多个,每个 Backend Node 之间通过高速的 LAN 或分布各地的 WAN 相连接。在实际的应用中,HAProxy Server 也可以同时兼任 Backend Node 的角色。

细心的读者可能已经发现了,在此架构中存在一个致命的缺点,从图 14-1 可以清楚地看出,所有的用户请求都经过 HAProxy Server 将任务分发到各个服务器节点,那么,当只有一台 HAProxy Server 时,将会出现单点故障点,一旦这个 HAProxy Server 出现故障,整个 HAProxy 系统将陷入瘫痪状态。

虽然 HAProxy Server 具有高负荷工作的能力,但是对于一个健壮的集群系统来说,单点故障是绝对不允许的。要避免这种单点故障,最实用、最简单的办法就是对 HAProxy Server 进行高可用集群,常见的方案就是为 HAProxy Server 做一个双机热备:在正常状态下主 HAProxy Server 工作,备用 HAProxy Server 监控主 HAProxy Server 的状态,当主 HAProxy Server 出现异常或者故障时,备用 HAProxy Server 马上接过主 HAProxy Server 的工作,负责对用户请求进行分发处理,这样就避免了一台 HAProxy Server 的单点故障问题,保证了负载均衡端持续地提供服务。

经过改进后的 HAProxy 负载均衡集群应用方案如图 14-2 所示。

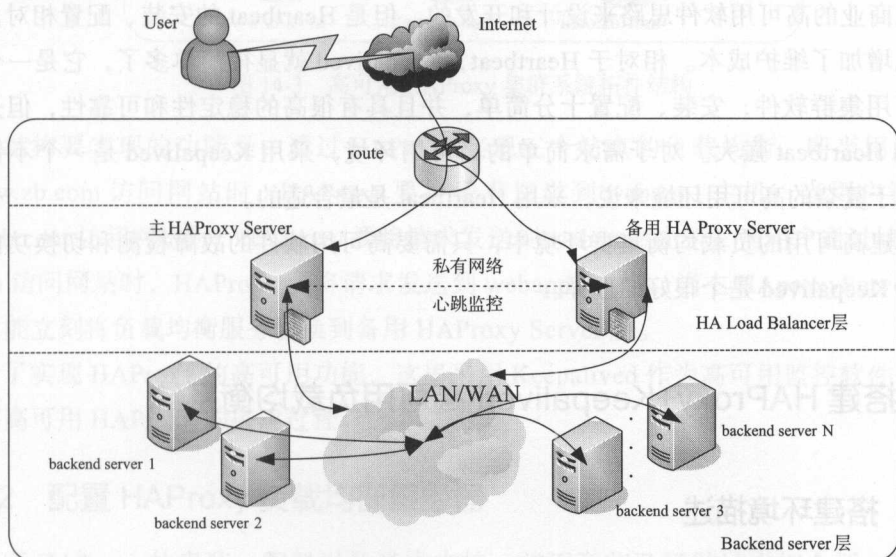


图 14-2 高可用的 HAProxy 负载均衡集群应用方案

从图 14-2 可以看出, 整个架构仍然分为三层, 在 HA Load Balancer 层, 主、备两台 HAProxy Server 构成双机热备系统, 双机之间通过心跳线连接。在正常状态下主 HAProxy Server 使用虚拟 IP 接收用户请求, 并根据设定好的策略和算法将请求分发给各个服务节点, 备用 HAProxy Server 负责监控主 HAProxy Server 的运行状态, 当主 HAProxy Server 发生异常或出现故障时, 备用 HAProxy Server 负责接管主 HAProxy Server 的虚拟 IP 和服务并继续接收用户请求和分发处理。通过这种相互监控策略, 任意一方主机故障对方都能够将 IP 和服务接管, 保证了 Load Balancer 层业务请求的不间断运行。

14.1.2 高可用集群软件的选择

为保证 HAProxy Server 不出现单点故障, 就需要通过高可用软件来实现, 目前开源的高可用软件有很多, 常用的有 Heartbeat 与 Keepalived。下面简单介绍下这两个软件的区别与联系。

Heartbeat 是 Linux-HA 项目中的一个组件, 也是目前开源 HA 项目中最成功的一个例子, 它提供了所有 HA 软件需要的基本功能, 比如心跳检测和资源接管, 监测集群中的系统服务, 在群集的节点间转移共享 IP 地址的所有者等。自 1999 年开始到现在, Heartbeat 在行业内得到了广泛应用, 也发行了很多的版本, 目前的稳定版本为 heartbeat3.0.5, 可以从 Linux-HA 的官方网站 www.linux-ha.org 下载所需的版本。

Keepalived 在前面的章节已经做过详细介绍, 它一方面具有服务器运行检测功能, 另一方面也具有 HA Cluster 功能, 目前很多高可用系统都是通过 Keepalived 来实现的。

综合来说, Heartbeat 是一个专业的 HA 软件, 最新版 Heartbeat 功能十分强大, 它完全是按照商业的高可用软件思路来设计和开发的, 但是 Heartbeat 的安装、配置相对比较复杂, 这也增加了维护成本。相对于 Heartbeat, Keepalived 就显得简单多了, 它是一个轻量级的高可用集群软件, 安装、配置十分简单, 并且具有很高的稳定性和可靠性, 但是它的功能不如 Heartbeat 强大。对于需求简单的高可用环境, 采用 Keepalived 是一个不错的选择, 而对于复杂的高可用环境来说, 采用 Heartbeat 是最合适的。

在构建高可用的负载均衡集群环境中, 只需要高可用软件的故障检测和切换功能, 因此, 采用 Keepalived 是个很好的选择。

14.2 搭建 HAProxy+Keepalived 高可用负载均衡系统

14.2.1 搭建环境描述

下面介绍如何通过 Keepalived 搭建高可用的 HAProxy 负载均衡集群系统, 在进行实例

介绍之前先进行约定：操作系统采用 CentOS6.3，地址规划如表 14.1 所示。

表 14-1 地址规划

主机名	IP 地址	集群角色	虚拟 IP
haproxy-server	192.168.66.11	主 HAProxyServer	192.168.66.10
backup-HAProxy	192.168.66.12	备用 HAProxyServer	
webapp1	192.168.66.20	Backend Server	无
webapp2	192.168.66.21		
webapp3	192.168.66.22		

整个高可用 HAProxy 集群系统的拓扑结构如图 14-3 所示。

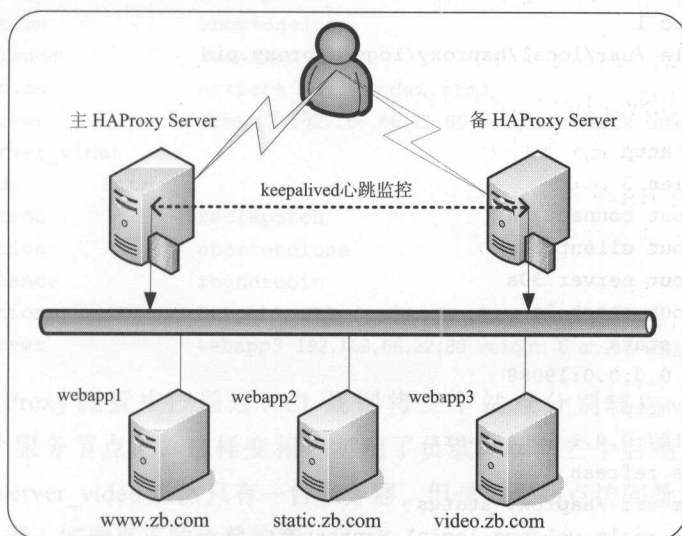


图 14-3 高可用 HAProxy 集群系统拓扑结构

此结构要实现的功能是：通过 HAProxy 实现三个站点的负载均衡，即当用户通过域名 `www.zb.com` 访问网站时，HAProxy 要将请求发送到 `webapp1` 主机；当用户通过域名 `static.zb.com` 访问网站时，HAProxy 要将请求发送到 `webapp2` 主机；当用户通过域名 `video.zb.com` 访问网站时，HAProxy 要将请求发送到 `webapp3` 主机；当主 HAProxyServer 发送故障后，能立刻将负载均衡服务切换到备用 HAProxy Server 上。

为了实现 HAProxy 的高可用功能，这里采用 Keepalived 作为高可用监控软件，下面依次介绍高可用 HAProxy 的搭建过程。

14.2.2 配置 HAProxy 负载均衡服务器

关于 HAProxy 的安装、配置以及日志支持，前面章节已经做过详细介绍，这里不再

重复介绍。首先在主、备 HAProxy 服务器上安装 HAProxy，并且配置日志支持，然后进入 HAProxy 的配置阶段，这里仅仅给出 HAProxy 的配置文件，主、备两个节点的 haproxy.conf 文件内容完全相同，这里假定 HAProxy 的安装路径为 /usr/local/haproxy。/usr/local/haproxy/conf/haproxy.conf 文件内容如下：

```
global
    log 127.0.0.1 local0 info
    maxconn 4096
    user nobody
    group nobody
    daemon
    nbproc 1
    pidfile /usr/local/haproxy/logs/haproxy.pid

defaults
    mode http
    retries 3
    timeout connect 5s
    timeout client 30s
    timeout server 30s
    timeout check 2s

listen admin_stats
    bind 0.0.0.0:19088
    mode http
    log 127.0.0.1 local0 err
    stats refresh 30s
    stats uri /haproxy-status
    stats realm welcome login\ Haproxy
    stats auth admin:xxxxxx
    stats hide-version
    stats admin if TRUE

frontend www
    bind 192.168.66.10:80
    mode http
    option httplog
    option forwardfor
    log global

    acl host_www          hdr_dom(host) -i      www.zb.com
    acl host_static        hdr_dom(host) -i      static.zb.com
    acl host_video         hdr_dom(host) -i      video.zb.com

    use_backend server_www if      host_www
    use_backend server_static if   host_static
```

```

        use_backend server_video if      host_video

backend server_www
    mode            http
    option          redispatch
    option          abortonclose
    balance         roundrobin
    option          httpchk GET /index.jsp
    server          webapp1 192.168.66.20:80 weight 6 check inter 2000 rise 2 fall 3

backend server_static
    mode            http
    option          redispatch
    option          abortonclose
    balance         roundrobin
    option          httpchk GET /index.html
    server          webapp2 192.168.66.21:80 weight 6 check inter 2000 rise 2 fall 3

backend server_video
    mode            http
    option          redispatch
    option          abortonclose
    balance         roundrobin
    option          httpchk GET /index.html
    server          webapp3 192.168.66.22:80 weight 6 check inter 2000 rise 2 fall 3

```

在这个 HAProxy 配置中，通过 ACL 规则将三个站点分别转向 webapp1、webapp2 和 webapp3 三个服务节点上，这样变相地实现了负载均衡。三个后端实例 server_www、server_static 和 server_video 虽然只有一台服务器，但是如果站点访问量增加，可以很容易地增加后端服务器，实现真正的负载均衡。

将 haproxy.conf 文件复制到备用的 backup-haproxy 服务器上，然后在主、备 HAProxy 上依次启动 HAProxy 服务。为了方便以后维护，最后将 HAProxy 的服务管理通过一个脚本来实现。HAProxy 管理脚本在第 12 章已经做过介绍，这里不再说明，将 HAProxy 管理脚本加到服务器自启动中，保证 HAProxy 服务开机就能运行。

14.2.3 配置主、备 Keepalived 服务器

依次在主、备两个节点上安装 Keepalived。关于 Keepalived 的安装这里不再介绍，直接给出配置好的 keepalived.conf 文件内容。在 haproxy-server 主机上，keepalived.conf 的内容如下：

```

global_defs {
    notification_email {
        acassen@firewall.loc

```

```

failover@firewall.loc
sysadmin@firewall.loc
}
notification_email_from Alexandre.Cassen@firewall.loc
smtp_server 192.168.200.1
smtp_connect_timeout 30
router_id HAProxy_DEVEL
}

vrrp_script check_haproxy {
script "killall -0 haproxy"          # 设置探测 HAProxy 服务运行状态的方式，这里的“killall
                                     #-0 haproxy”仅仅是检测 HAProxy 服务状态

interval 2
weight 21
}

vrrp_instance HAProxy_HA {
state BACKUP                        # 在 haproxy-server 和 backup-haproxy 上均配置为 BACKUP
interface eth0
virtual_router_id 80
priority 100
advert_int 2
nopreempt                          # 不抢占模式，只在优先级高的机器上设置即可，优先级低的机器可不设置
authentication {
auth_type PASS
auth_pass 1111
}

notify_master "/etc/keepalived/mail_notify.py master"
notify_backup "/etc/keepalived/mail_notify.py backup"
notify_fault "/etc/keepalived/mail_notify.py falut"

track_script {
check_haproxy
}

virtual_ipaddress {
192.168.66.10/24 dev eth0 #HAProxy 的对外服务 IP，即 VIP
}
}

```

其中，`/etc/keepalived/mail_notify.py` 文件是一个邮件通知程序，当 Keepalived 进行 Master、Backup、Fault 状态切换时，将会发送通知邮件给运维人员，这样可以及时了解高可用集群的运行状态，以便在适当的时候人为介入处理故障。`mail_notify.py` 文件的内容如下：

```

#!/usr/bin/env python
# -*- coding: utf-8 -*-
import sys
reload(sys)
from email.MIMEText import MIMEText
import smtplib
import MySQLdb
sys.setdefaultencoding('utf-8')
import socket, fcntl, struct

def send_mail(to_list, sub, content):
    mail_host="smtp.163.com"      # 设置验证服务器，这里以 163.com 为例
    mail_user="username"         # 设置验证用户名
    mail_pass="xxxxxx"          # 设置验证口令
    mail_postfix="163.com"       # 设置邮箱的后缀
    me=mail_user+"<" + mail_user + "@" + mail_postfix + ">"
    msg = MIMEText(content)
    msg['Subject'] = sub
    msg['From'] = me
    msg['To'] = to_list
    try:
        s = smtplib.SMTP()
        s.connect(mail_host)
        s.login(mail_user, mail_pass)
        s.sendmail(me, to_list, msg.as_string())
        s.close()
        return True
    except Exception, e:
        print str(e)
        return False

def get_local_ip(iframe = 'eth0'):
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    inet = fcntl.ioctl(s.fileno(), 0x8915, struct.pack('256s', iframe[:15]))
    ret = socket.inet_ntoa(inet[20:24])
    return ret

if sys.argv[1]!="master" and sys.argv[1]!="backup" and sys.argv[1]!="fault":
    sys.exit()
else:
    notify_type = sys.argv[1]

if __name__ == '__main__':
    strcontent = get_local_ip() + " " + notify_type + " 状态被激活，请确认 HAProxy 服务运行状态！"
    # 下面这段是设置接收报警信息的邮件地址列表，可设置多个
    mailto_list = ['xxxxxx@163.com', 'xxxxxx@qq.com']

```



```
for mailto in mailto_list:
    send_mail(mailto, "HAProxy 状态切换报警", strcontent.encode('utf-8'))
```

最后，将 `keepalived.conf` 文件和 `mail_notify.py` 文件复制到 `backup-haproxy` 服务器上对应的位置，然后将 `keepalived.conf` 文件中 `priority` 值修改为 90，由于配置的是不抢占模式，因此，还需要在 `backup-haproxy` 服务器上去掉 `nopreempt` 选项。

完成所有配置后，分别在 `haproxy-server` 和 `backup-haproxy` 主机上依次启动 HAProxy 服务和 Keepalived 服务。注意，这里一定要先启动 HAProxy 服务，因为 Keepalived 服务在启动的时候会自动检测 HAProxy 服务是否正常，如果发现 HAProxy 服务没有启动，那么主、备 Keepalived 将自动进入 Fault 状态。在依次启动服务后，在正常情况下 VIP 地址应该运行在 `haproxy-server` 服务器上，通过命令 “`ip a`” 可以查看 VIP 是否已经正常加载。

14.3 测试 HAProxy+Keepalived 高可用负载均衡集群

高可用的 HAProxy 负载均衡系统能够实现 HAProxy 的高可用性、负载均衡特性和故障自动切换特性。由于本章介绍的高可用构架只涉及高可用性和负载均衡两个特性，因此，对其进行的测试仅针对这两个方面进行。下面进行简单的测试。

14.3.1 测试 Keepalived 的高可用功能

高可用性是通过 HAProxy 的两个 HAProxy Server 完成的。为了模拟故障，先将主 `haproxy-server` 上面的 HAProxy 服务停止，接着观察 `haproxy-server` 上 Keepalived 的运行日志，信息如下：

```
Apr  4 20:57:51 haproxy-server Keepalived_vrrp[23824]: VRRP_Script(check_haproxy) failed
Apr  4 20:57:54 haproxy-server Keepalived_vrrp[23824]: VRRP_Instance(HA_1)
Received higher prio advert
Apr  4 20:57:54 haproxy-server Keepalived_vrrp[23824]: VRRP_Instance(HA_1)
Entering BACKUP STATE
Apr  4 20:57:54 haproxy-server Keepalived_vrrp[23824]: VRRP_Instance(HA_1)
removing protocol VIPs.
Apr  4 20:57:54 haproxy-server Keepalived_healthcheckers[23823]: Netlink reflector
reports IP 192.168.66.10 removed
```

这段日志显示了 `check_haproxy` 检测失败后，`haproxy-server` 自动进入了 BACKUP 状态，同时释放了虚拟 IP。由于执行了角色切换，此时 `mail_notify.py` 脚本应该会自动执行并发送状态切换邮件，类似的邮件信息如图 14-4 所示。

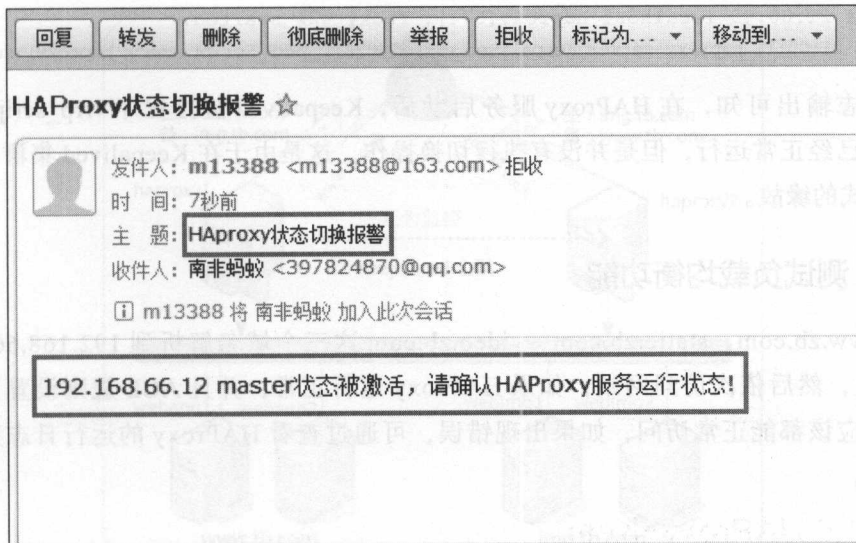


图 14-4 Keepalived 状态切换时的报警邮件

然后观察备机 backup-haproxy 上 Keepalived 的运行日志, 信息如下:

```
Apr  4 20:57:54 backup-haproxy Keepalived_vrrp[17261]: VRRP_Instance(HA_1) forcing
a new MASTER election
Apr  4 20:57:54 backup-haproxy Keepalived_vrrp[17261]: VRRP_Instance(HA_1) forcing
a new MASTER election
Apr  4 20:57:56 backup-haproxy Keepalived_vrrp[17261]: VRRP_Instance(HA_1)
Transition to MASTER STATE
Apr  4 20:57:58 backup-haproxy Keepalived_vrrp[17261]: VRRP_Instance(HA_1)
Entering MASTER STATE
Apr  4 20:57:58 backup-haproxy Keepalived_vrrp[17261]: VRRP_Instance(HA_1) setting
protocol VIPs.
Apr  4 20:57:58 backup-haproxy Keepalived_healthcheckers[17260]: Netlink reflector
reports IP 192.168.66.10 added
Apr  4 20:57:58 backup-haproxy avahi-daemon[1207]: Registering new address record
for 192.168.66.10 on eth0.IPv4.
Apr  4 20:57:58 backup-haproxy Keepalived_vrrp[17261]: VRRP_Instance(HA_1) Sending
gratuitous ARPs on eth0 for 192.168.66.10
Apr  4 20:58:03 backup-haproxy Keepalived_vrrp[17261]: VRRP_Instance(HA_1) Sending
gratuitous ARPs on eth0 for 192.168.66.10
```

从日志中可以看出, 主机出现故障后, backup-haproxy 立刻检测到, 此时 backup-haproxy 变为 Master 角色, 并且接管了主机的虚拟 IP 资源, 最后将虚拟 IP 绑定在 eth0 设备上。

接着, 重新启动主 haproxy-server 上的 Keepalived 服务, 然后观察 haproxy-server 上的日志状态:

```
Apr  4 21:00:09 localhost haproxy[574]: Proxy www started.
Apr  4 21:00:11 haproxy-server Keepalived_vrrp[23824]: VRRP_Script(check_haproxy) succeeded
```

从日志输出可知，在 HAProxy 服务启动后，Keepalived 监控程序 vrrp_script 检测到 HAProxy 已经正常运行，但是并没有执行切换操作，这是由于在 Keepalived 集群中设置了不抢占模式的缘故。

14.3.2 测试负载均衡功能

将 www.zb.com、static.zb.com、video.zb.com 这三个域名解析到 192.168.66.10 这个虚拟 IP 上，然后依次访问网站，如果 HAProxy 运行正常，并且 ACL 规则设置正确，这三个网站应该都能正常访问，如果出现错误，可通过查看 HAProxy 的运行日志判断哪里出了问题。

14.4 构建双主高可用的 HAProxy 负载均衡系统

在上面介绍的 HAProxy 高可用负载均衡集群架构中，虽然通过 Keepalived 实现了 HAProxy 的高可用，但是严重浪费了服务器资源，因为在一主一备的 Keepalived 环境中，只有主节点处于工作状态，而备用节点则一直处于空闲等待状态，仅当主节点出现问题时备用节点才能开始工作。对于并发量比较大的 Web 应用系统来说，主节点可能会非常繁忙，而备用节点则十分空闲，这种服务器资源分布不均的问题，也是在做应用架构设计时必须要考虑的问题。对于一主一备资源不均衡的问题，可以通过双主互备的方式进行负载分流，下面就详细讲述双主互备的高可用集群系统是如何实现的。

14.4.1 系统架构图与实现原理

为了能充分利用服务器资源并将负载进行分流，可以在一主一备的基础上构建双主互备的高可用 HAProxy 负载均衡集群系统。双主互备的集群架构如图 14-5 所示。

在这个架构中，要实现的功能是：通过 haproxy1 服务器将 www.tb.com 的访问请求发送到 webapp1 和 webapp2 两台主机上，实现 www.tb.com 的负载均衡；通过 haproxy2 将 img.tb.com 的访问请求发送到 webimg1 和 webimg2 两台主机上，实现 img.tb.com 的负载均衡；同时，如果 haproxy1 或 haproxy2 任何一台服务器出现故障，都会将用户访问请求发送到另一台健康的负载均衡节点，进而继续保持两个网站的负载均衡。

在进行实例介绍之前进行约定：操作系统采用 CentOS6.3，地址规划如表 14-2 所示。

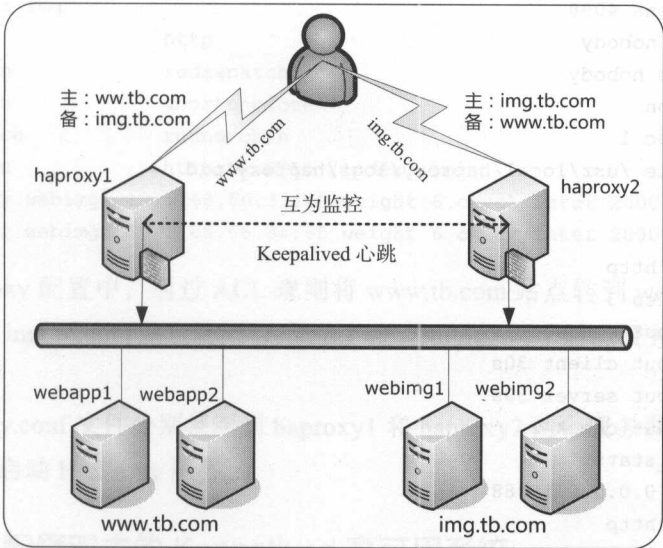


图 14-5 HAProxy 双主互备集群架构

表 14-2 双主高可用负载均衡地址规划

主机名	物理 IP 地址	虚拟 IP 地址	集群角色
haproxy1	192.168.66.11	192.168.66.10	主: www.tb.com
			备: img.tb.com
haproxy2	192.168.66.12	192.168.66.20	主: img.tb.com
			备: www.tb.com
webapp1	192.168.66.31	无	Backend Server
webapp2	192.168.66.32		Backend Server
webimg1	192.168.66.33	无	Backend Server
webimg2	192.168.66.34		Backend Server

注意，为了保证 haproxy1 和 haproxy2 服务器资源得到充分利用，这里对访问进行了分流操作，需要将 www.tb.com 的域名解析到 192.168.66.10 这个 IP 上，将 img.tb.com 域名解析到 192.168.66.20 这个 IP 上。

14.4.2 安装并配置 HAProxy 集群系统

在主机名为 haproxy1 和 haproxy2 的节点依次安装 HAProxy 并配置日志支持，在此略过这个过程。HAProxy 的安装路径仍然指定为 /usr/local/haproxy，两个节点的 haproxy.cfg 文件内容完全相同，这里给出配置好的 haproxy.cfg 文件，内容如下：

```
global
    log 127.0.0.1 local0 info
```



```

maxconn 4096
user nobody
group nobody
daemon
nbproc 1
pidfile /usr/local/haproxy/logs/haproxy.pid

```

defaults

```

mode http
retries 3
timeout connect 5s
timeout client 30s
timeout server 30s
timeout check 2s

listen admin_stats
bind 0.0.0.0:19088
mode http
log 127.0.0.1 local0 err
stats refresh 30s
stats uri /haproxy-status
stats realm welcome login\ Haproxy
stats auth admin:xxxxxx
stats hide-version
stats admin if TRUE

```

frontend www

```

bind *:80
mode http
option httplog
option forwardfor
log global

```

```

acl host_www      hdr_dom(host) -i www.tb.com
acl host_img      hdr_dom(host) -i img.tb.com

```

```

use_backend server_www if host_www
use_backend server_img if host_img

```

backend server_www

```

mode http
option redispatch
option abortonclose
balance roundrobin
option httpchk HEAD /index.php
server webapp1 192.168.66.31:80 weight 6 check inter 2000 rise 2 fall 3
server webapp2 192.168.66.32:80 weight 6 check inter 2000 rise 2 fall 3

```

```

backend server_img
    mode                http
    option               redispatch
    option               abortonclose
    balance              roundrobin
    option               httpchk HEAD /index.html
    server webimg1 192.168.66.33:80 weight 6 check inter 2000 rise 2 fall 3
    server webimg2 192.168.66.34:80 weight 6 check inter 2000 rise 2 fall 3

```

在这个 HAProxy 配置中, 通过 ACL 规则将 www.tb.com 站点转到 webapp1、webapp2 两个后端节点, 将 img.tb.com 站点转到 webimg1 和 webimg2 两个后端服务节点, 分别实现负载均衡。

最后将 haproxy.conf 文件分别复制到 haproxy1 和 haproxy2 两台服务器上, 然后在两个负载均衡器上依次启动 HAProxy 服务。

14.4.3 安装并配置双主的 Keepalived 高可用系统

依次在主、备两个节点上安装 Keepalived。关于 Keepalived 的安装这里略过, 在 haproxy1 主机和 haproxy2 主机上, keepalived.conf 的内容基本相同, 不同的部分已经在后面添加了注释, 下面直接给出配置好的 keepalived.conf 文件内容:

```

global_defs {
    notification_email {
        acassen@firewall.loc
        failover@firewall.loc
        sysadmin@firewall.loc
    }
    notification_email_from Alexandre.Cassen@firewall.loc
    smtp_server 192.168.200.1
    smtp_connect_timeout 30
    router_id haproxy_DEVEL
}

vrrp_script check_haproxy {
    script "killall -0 haproxy"
    interval 2
}

vrrp_instance haproxy_hal {
    state MASTER # 在 haproxy2 主机上, 此处为 BACKUP
    interface eth0
    virtual_router_id 80 # 在一个实例下, virtual_router_id 是唯一的, 因此在 haproxy2 主
                        # 机上, 此处 virtual_router_id 也必须为 80
    priority 100 # 在 haproxy2 主机上, priority 值为 80
}

```

```

advert_int 2
authentication {
    auth_type PASS
    auth_pass 1111
}
notify_master "/etc/keepalived/mail_notify.py master"
notify_backup "/etc/keepalived/mail_notify.py backup"
notify_fault "/etc/keepalived/mail_notify.py fault"

track_script {
    check_haproxy
}

virtual_ipaddress {
    192.168.66.10/24 dev eth0
}

vrrp_instance haproxy_ha2 {
    state BACKUP # 在 haproxy2 主机上, 此处为 MASTER
    interface eth0
    virtual_router_id 81 # 在 haproxy2 主机上, 此处 virtual_router_id 也必须为 81
    priority 80 # 在 haproxy2 主机上, priority 值为 100
    advert_int 2
    authentication {
        auth_type PASS
        auth_pass 1111
    }

    notify_master "/etc/keepalived/mail_notify.py master"
    notify_backup "/etc/keepalived/mail_notify.py backup"
    notify_fault "/etc/keepalived/mail_notify.py fault"

    track_script {
        check_haproxy
    }
    virtual_ipaddress {
        192.168.66.20/24 dev eth0
    }
}

```

在双主互备的配置中, 有两个 VIP 地址, 在正常情况下, 192.168.66.10 将自动加载到 haproxy1 主机上, 而 192.168.66.20 将自动加载到 haproxy2 主机上。这里要特别注意的是, haproxy1 和 haproxy2 两个节点上 virtual_router_id 的值要互不相同, 并且 MASTER 角色的 priority 值要大于 BACKUP 角色的 priority 值。

在完成所有配置修改后,依次在 haproxy1 和 haproxy2 两个节点启动 Keepalived 服务,并观察 VIP 地址是否正常加载到对应的节点上。

14.4.4 测试双主高可用的 HAProxy 负载均衡集群系统

关于负载均衡功能的测试前面章节已经介绍过多次,这里仅测试双主互备的 HAProxy 负载均衡系统的启动和高可用切换过程。

在 haproxy1 和 haproxy2 节点依次启动 HAProxy 服务和 Keepalived 服务后,首先观察 haproxy1 节点 Keepalived 的启动日志,信息如下:

```
Apr  5 01:16:22 haproxy1 Keepalived_healthcheckers[29104]: Opening file '/etc/keepalived/keepalived.conf'.
Apr  5 01:16:22 haproxy1 Keepalived_healthcheckers[29104]: Configuration is using : 7818 Bytes
Apr  5 01:16:22 haproxy1 Keepalived_vrrp[29105]: VRRP_Instance(HAProxy_HA2) Entering BACKUP STATE
Apr  5 01:16:22 haproxy1 Keepalived_healthcheckers[29104]: Using LinkWatch kernel netlink reflector...
Apr  5 01:16:22 haproxy1 Keepalived_vrrp[29105]: VRRP sockpool: [ifindex(2),proto(112),unicast(0), fd(10,11)]
Apr  5 01:16:22 haproxy1 Keepalived_vrrp[29105]: VRRP_Script(check_haproxy) succeeded
Apr  5 01:16:24 haproxy1 Keepalived_vrrp[29105]: VRRP_Instance(HAProxy_HA1) Transition to MASTER STATE
Apr  5 01:16:26 haproxy1 Keepalived_vrrp[29105]: VRRP_Instance(HAProxy_HA1) Entering MASTER STATE
Apr  5 01:16:26 haproxy1 Keepalived_vrrp[29105]: VRRP_Instance(HAProxy_HA1) setting protocol VIPs.
Apr  5 01:16:26 haproxy1 Keepalived_vrrp[29105]: VRRP_Instance(HAProxy_HA1) Sending gratuitous ARPs on eth0 for 192.168.66.10
Apr  5 01:16:26 haproxy1 Keepalived_healthcheckers[29104]: Netlink reflector reports IP 192.168.66.10 added
Apr  5 01:16:27 haproxy1 ntpd[29393]: Listening on interface #65 eth0, 192.168.66.10#123 Enabled
Apr  5 01:16:31 haproxy1 Keepalived_vrrp[29105]: VRRP_Instance(HAProxy_HA1) Sending gratuitous ARPs on eth0 for 192.168.66.10
```

从输出可知,Keepalived 启动了 HAProxy_HA1 和 HAProxy_HA2 两个实例,在检测到 HAProxy 服务运行正常后,HAProxy_HA1 实例进入 MASTER 状态,而 HAProxy_HA2 自动进入 BACKUP 状态。

接着在 haproxy2 节点查看 Keepalived 的启动日志,信息如下:

```
Apr  5 01:16:24 haproxy2 Keepalived_vrrp[24550]: Opening file '/etc/keepalived/keepalived.conf'.
```



```

Apr  5 01:16:24 haproxy2 Keepalived_vrrp[24550]: Configuration is using : 70726 Bytes
Apr  5 01:16:24 haproxy2 Keepalived_vrrp[24550]: Using LinkWatch kernel netlink
reflector...
Apr  5 01:16:24 haproxy2 Keepalived_vrrp[24550]: VRRP_Instance(HAProxy_HA1)
Entering BACKUP STATE
Apr  5 01:16:24 haproxy2 Keepalived_vrrp[24550]: VRRP sockpool: [ifindex(2),
proto(112), unicast(0), fd(10,11)]
Apr  5 01:16:24 haproxy2 Keepalived_healthcheckers[24549]: Using LinkWatch kernel
netlink reflector...
Apr  5 01:16:24 haproxy2 Keepalived_vrrp[24550]: VRRP_Script(check_haproxy) succeeded
Apr  5 01:16:26 haproxy2 Keepalived_vrrp[24550]: VRRP_Instance(HAProxy_HA2)
Transition to MASTER STATE
Apr  5 01:16:28 haproxy2 Keepalived_vrrp[24550]: VRRP_Instance(HAProxy_HA2)
Entering MASTER STATE
Apr  5 01:16:28 haproxy2 Keepalived_vrrp[24550]: VRRP_Instance(HAProxy_HA2)
setting protocol VIPs.
Apr  5 01:16:28 haproxy2 Keepalived_vrrp[24550]: VRRP_Instance(HAProxy_HA2)
Sending gratuitous ARPs on eth0 for 192.168.66.20
Apr  5 01:16:28 haproxy2 Keepalived_healthcheckers[24549]: Netlink reflector
reports IP 192.168.66.20 added
Apr  5 01:16:28 haproxy2 avahi-daemon[1207]: Registering new address record for
192.168.66.20 on eth0.IPv4.
Apr  5 01:16:33 haproxy2 Keepalived_vrrp[24550]: VRRP_Instance(HAProxy_HA2)
Sending gratuitous ARPs on eth0 for 192.168.66.20

```

从输出可知, haproxy2 节点的 Keepalived 也启动了 HAProxy_HA1 和 HAProxy_HA2 两个实例, 在检测到 HAProxy 服务运行正常后, HAProxy_HA1 实例进入 BACKUP 状态, 而 HAProxy_HA2 自动进入 MASTER 状态。

通过查看 haproxy1 和 haproxy2 中 Keepalived 日志的运行状态, 可以发现完全符合双主互备高可用集群的运行原理。

下面测试一下双主互备的故障切换功能, 这里为模拟故障, 将 haproxy1 节点上 HAProxy 服务关闭, 然后在 haproxy1 节点观察 Keepalived 的启动日志, 信息如下:

```

Apr  5 01:19:16 haproxy1 Keepalived_vrrp[29105]: VRRP_Script(check_haproxy) failed
Apr  5 01:19:18 haproxy1 Keepalived_vrrp[29105]: VRRP_Instance(HAProxy_HA1)
Entering FAULT STATE
Apr  5 01:19:18 haproxy1 Keepalived_vrrp[29105]: VRRP_Instance(HAProxy_HA1)
removing protocol VIPs.
Apr  5 01:19:18 haproxy1 Keepalived_vrrp[29105]: VRRP_Instance(HAProxy_HA1) Now
in FAULT state
Apr  5 01:19:18 haproxy1 Keepalived_healthcheckers[29104]: Netlink reflector
reports IP 192.168.66.10 removed

```

从输出日志可知, HAProxy_HA1 实例在检测 HAProxy 服务失败后, 自动进入 FAULT

状态，同时释放了 VIP 地址 192.168.66.10。

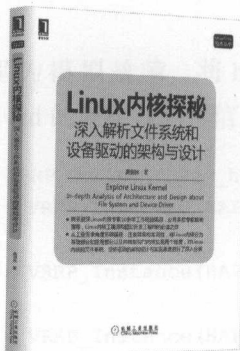
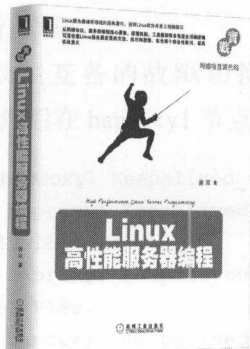
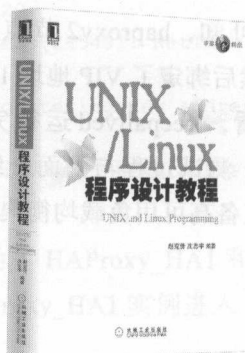
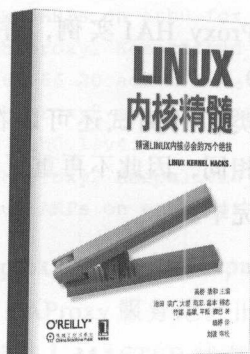
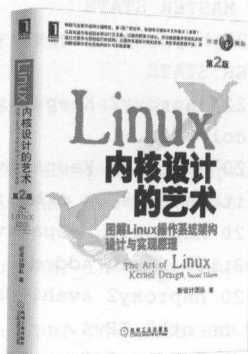
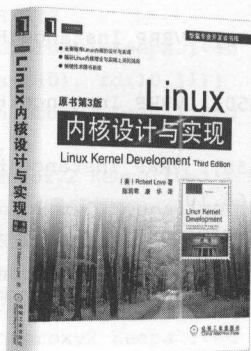
接着查看下 haproxy2 节点中 Keepalived 日志的运行状态，信息如下：

```
Apr  5 01:19:18 haproxy2 Keepalived_vrrp[24550]: VRRP_Instance(HAProxy_HA1)
Transition to MASTER STATE
Apr  5 01:19:20 haproxy2 Keepalived_vrrp[24550]: VRRP_Instance(HAProxy_HA1)
Entering MASTER STATE
Apr  5 01:19:20 haproxy2 Keepalived_vrrp[24550]: VRRP_Instance(HAProxy_HA1)
setting protocol VIPs.
Apr  5 01:19:20 haproxy2 Keepalived_vrrp[24550]: VRRP_Instance(HAProxy_HA1)
Sending gratuitous ARPs on eth0 for 192.168.66.10
Apr  5 01:19:20 haproxy2 Keepalived_healthcheckers[24549]: Netlink reflector
reports IP 192.168.66.10 added
Apr  5 01:19:20 haproxy2 avahi-daemon[1207]: Registering new address record for
192.168.66.10 on eth0.IPv4.
Apr  5 01:19:25 haproxy2 Keepalived_vrrp[24550]: VRRP_Instance(HAProxy_HA1)
Sending gratuitous ARPs on eth0 for 192.168.66.10
```

从输出日志可知，haproxy2 节点接管了 HAProxy_HA1 实例，并将此实例切换到 MASTER 状态，然后绑定了 VIP 地址 192.168.66.10。

从切换过程看，Keepalived 运行完全正常。类似的测试还可以在 haproxy2 节点关闭 HAProxy 服务，测试过程与上面介绍过的完全相同，因此不再重复讲述。至此，基于 HAProxy 的双主互备高可用负载均衡集群方案讲解完毕。

推荐阅读



性能和稳定性一直是服务器（尤其是面向大规模服务器集群）运维领域最重要的主题，也是运维工程师公认最难的主题之一，因为影响服务器性能和稳定性的因素太多，而且问题会随着业务需求、服务器规模、服务器运行环境和运维技术的变化而不断变化。“高性能Linux服务器构建实战”系列图书以具有一定规模的Linux服务器集群为研究对象，以帮助运维工程师构建高可靠、高性能的Linux服务器为目标，逐一对影响服务器性能的各种因素和能提高服务器性能的各种方法抽丝剥茧，深入分析。

《高性能Linux服务器构建实战：运维监控、性能调优与集群应用》一书从Web应用系统、数据备份恢复、网络存储、运维监控、性能优化、集群应用等角度切入，上市后在运维领域引起了广泛的关注和强烈的反响，读者给予了极高的评价，被誉为Linux服务器运维领域少有的经典著作之一。本书与《高性能Linux服务器构建实战：运维监控、性能调优与集群应用》一脉相承、互为补充，从系统安全、故障排查、自动化运维、集群架构4个不同的维度讲解了构建大规模和高性能Linux服务器集群所需技术、工具、方法及技巧。



投稿热线: (010) 88379604
客服热线: (010) 88379426 88361066
购书热线: (010) 68326294 88379649 68995259

华章网站: www.hzbook.com
网上购书: www.china-pub.com
数字阅读: www.hzmedia.com.cn

上架指导: 计算机/操作系统

ISBN 978-7-111-47249-0



9 787111 472490 >

定价: 79.00元